(2)

FINAL TECHNICAL REPORT TO

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH

by

Jeffery L. Kennington
Department of Computer Science and Engineering
Southern Methodist University
Dallas, TX 75275-0122
(214)-692-3278

for

Optimization Algorithms for New Computer Architectures With Applications to Routing and Scheduling

February 20, 1992

AFOSR 91-0028

SMU # 5-25134

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188
1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION SMU	6b. OFFICE SYMBOL (If applicable) CSE	7a. NAME OF MONITORING ORGANIZATION		
6c. ADDRESS (City, State, and ZIP Code) Dallas, TX 75275-0211		7b. ADDRESS (City, State, and ZIP Code)		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL (If applicable) NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code) Building 410 Bolling AFB, DC 20332-6448		10. SOURCE OF FUNDING NUMBERS		
		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) [illegible]

12. PERSONAL AUTHOR(S) Jeffery L. Kennington

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 01Oct 90 TO 31Dec 91	02 Feb 1992	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document presents new serial and parallel algorithms for various network flow models. The models investigated include the one-to-one shortest path problem, sparse and dense assignment problems, the transportation problem, the singly constrained assignment problem, and the generalized network problem. Algorithms for all of these models have been developed and empirically tested on a variety of sequential and parallel computers. One-million arc test problems have been successfully solved with several of the new algorithms.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT		21. ABSTRACT SECURITY CLASSIFICATION
[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS		Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Jeffery L. Kennington	22b. TELEPHONE (Include Area Code) (214) 692-3278	22c. OFFICE SYMBOL SEAS

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

Table of Contents

I. STATEMENT OF WORK

Many Air Force applications can be modeled as some extension of a pure network problem. These extensions may require additional side constraints, arcs that involve attrition or flow of multiple commodities on a single arc. In all cases, the network models require integer flows and may be viewed as special cases of the integer programming model. Since some of these applications demand computer hardware several orders of magnitude faster than the fastest machines available, we have investigated the use of parallelism to increase the computational speed of these algorithms. Very powerful hardware (in terms of millions of floating point operations per second) can be built using many low cost standard chips, all designed to operate in parallel. Our research program objective is to develop and empirically test new serial and parallel algorithms and software for network based models. The problems studied during the past eighteen months include the generalized network problem, the transportation problem, sparse and dense assignment problems, the one–to–one shortest path problem problem, and the singly constrained assignment problem. Algorithms for all of these models have been developed and empirically tested on a variety of sequential and parallel computers.

II. PUBLICATIONS

Title

An Empirical Analysis of the Dense Assignment Problem: Sequential and Parallel Implementations

Revised May 1991

Authors

J. Kennington and Z. Wang

Executive Summary

We performed a thorough empirical analysis comparing the auction algorithm with the shortest augmenting path algorithm (SAP) for the dense assignment problem. We found that the software implementation of the SAP algorithm was superior on serial machines. A parallel implementation of this software yielded speedups of four using ten processors. We successfully solved problems having over one million arcs in less that 20 seconds on a Sequent Symmetry S81.

Publication Status

This paper appears in the ORSA Journal on Computing, 3, 299–306, 1991.

Title

The Singly Constrained Assignment Problem

Revised December 1991

Authors

J. Kennington and F. Mohammadi

Executive Summary

This paper presents a new algorithm for the singly constrained assignment problem along with an empirical analysis of the software implementation of this algorithm. The new algorithm is based on Lagrangean duality theory and involves solving a series of pure assignment problems. The software implementation has successfully solved problems having over one–half million binary variables in less than fifteen minutes on a Sequent Symmetry S81 using a single processor.

Publication Status

This paper has been submitted for publication and is currently under review.

Title

Generalized Networks: Parallel Algorithms and an Empirical Analysis

Revised January 1991

Authors

R. Clark, J. Kennington, R. Meyer and M. Ramamurti

Executive Summary

A generalized network problem is a specialization of the linear programming problem in which each column of the constraint matrix has at most two nonzero entries. It is well known that a generalized network problem possesses a special graphical structure which can be exploited in algorithm development. Specialized sequential and parallel codes which exploit this graphical structure have been developed and empirically tested on a Sequent Symmetry S81.

Publication Status

This paper has been accepted for publication in the ORSA Journal on Computing.

Title

Computational Comparison of Sequential and Parallel Algorithms for the One–To–One Shortest–Path Problem

Revised January 1992

Authors

R. Helgason, J. Kennington, and B. Stewart

Executive Summary

The problem of finding the shortest path between a designated pair of nodes is a fundamental problem in operations research which also serves as a building block for other algorithms. The classical Dijkstra algorithm begins at one of the designated nodes and fans out from this node until the other designated node becomes a member of the labeled set. In this paper we empirically demonstrate that a better algorithm is obtained by a procedure that begins at both designated nodes and fans out in both directions either simultaneously in parallel or alternately in series.

Publication Status

This paper has been submitted for publication and is currently under review.

Title

The Shortest Augmenting Path Algorithm for the Transportation Problem

Revised February 1991

Authors

J. Kennington and Z. Wang

Executive Summary

This study presents an empirical analysis of the shortest augmenting path algorithm, augmented by advanced start heuristics, for the transportation problem. The software implementation of our algorithm is the best available for problems having a small total supply. As the total supply increases, the algorithm degrades and is computationally slower than competing primal simplex software.

Publication Status

This paper has been issued as a technical report.

Technical Report 88-OR-16

AN EMPIRICAL ANALYSIS OF THE DENSE ASSIGNMENT PROBLEM: SEQUENTIAL AND PARALLEL IMPLEMENTATIONS

by

Jeffery L. Kennington
Department of Computer Science & Engineering
Southern Methodist University
Dallas, Texas 75275-0122
BITNET: E4CR1001@SMUVM1
FAX: (214)-692-4138

Zhiming Wang
American Airlines Decision Technologies
P. O. Box 619616\MD 3345
DFW Airport, Texas 75261-9616
FAX: (817)-967-9763

revised May 1991

Has appeared in the ORSA Journal on Computing,3, 299-306, 1991.

ABSTRACT

The best algorithms for the dense assignment problem are acknowledged to be the auction algorithm and the shortest augmenting path algorithm. In this investigation we present an empirical analysis of two of the current best software implementations of these two methods on three different serial machines. These software implementations were developed by Bertsekas of the Massachusetts Institute of Technology and by Jonker and Volgenant of the University of Amsterdam. This report is an independent evaluation of the software implementation of these two algorithms. For the sample of problems examined and the sample of hardware used (IBM 3081D, Sequent Symmetry S81, and VAX 750), we found that the shortest augmenting path algorithm was the fastest. We also report our empirical results with a parallel version of the shortest augmenting path algorithm. On 1200x1200 dense assignment problems, speedups of approximately four were achieved using ten processors. Million arc problems were solved in less than twelve seconds on a Sequent Symmetry S81 with the parallel shortest augmenting path algorithm.

ACKNOWLEDGMENT

The authors wish to express their appreciation to Dimitri P. Bertsekas and Paul Tseng of the Massachusetts Institute of Technology for providing invaluable assistance during this study and to Roy Jonker and Ton Volgenant of the University of Amsterdam for sharing their software with us. This research was supported in part by the Department of Defense under Contract Number MDA 903-86-C0182, the Air Force Office of Scientific Research under Contract Number AFOSR 87-0199, the Office of Naval Research under Contract Number N00014-87-K-0223, and ROME Air Development Center under Contract Number SCEEE PDP/87-95. This manuscript has benefited from a careful reading by Richard V. Helgason of Southern Methodist University.

The classical assignment problem (also known as the weighted bipartite matching problem) is to assign n men to n distinct jobs so that the total cost of as-·-̧nment is minimized. Mathematically, this may be formulated as the following special mathematical program:

$$\begin{aligned} \text{minimize} \quad & \sum_{i,j} c_{ij} \; x_{ij} \\ \text{subject to:} \quad & \sum_{i} \; x_{ij} = 1, \;\; j = 1, \ldots, n \\ & \sum_{j} \; x_{ij} = 1, \;\; i = 1, \ldots, n \\ & x_{ij} \in \{0, 1\}, \;\; (\text{all } i, j) \end{aligned}$$

where c_{ij} denotes the cost for assigning man i to job j and $x_{ij} = 1$ implies that man i is assigned to job j. Due to the total unimodularity of the constraint matrix, this problem can be solved by the simplex algorithm and every basic solution will have $x_{ij} \in \{0, 1\}$ (see Kennington and Helgason [1980]). Hence, classic linear programming duality theory and Kuhn–Tucker optimality conditions can be used in algorithm development for this problem.

Specialized algorithms for the assignment problem can be classified into five categories as follows:

(i) maximum flow (primal-dual),

(ii) primal simplex,

(iii) dual simplex,

(iv) auction algorithm, and

(v) shortest augmenting paths.

The first maximum flow algorithm was developed by Kuhn [1955] and is called the Hungarian method. Its name comes from the fact that the algorithm is developed from results of two Hungarian mathematicians. Variations were presented by Kuhn [1956]. Derigs [1985] shows that the shortest augmenting path method can be viewed as a special implementation of the Hungarian method. Ahuja, Magnanti, and Orlin [1989] call the Hungarian method a primal-dual variant of the successive augmenting path algorithm.

A specialized primal simplex algorithm for the assignment problem was developed by Barr, Glover and Klingman [1977]. Their method, called the alternating basis algorithm, only considers a subset of the possible bases. This idea was further exploited by Hung [1983] in his development of a polynomial simplex algorithm for this problem. An extensive computational study comparing the Hungarian algorithm with primal simplex methods was performed by McGinnis [1983].

A dual polynomial simplex method known as the signature method has been developed by Balinski [1985, 1986]. Extensions for the sparse assignment problem were developed by Goldfarb [1985], and the relationship between the signature method and the shortest augmenting path method was presented by Derigs [1985]. Another variation of this algorithm has been developed by Akgul [1988].

Bertsekas [1979, 1981, 1988] and Bertsekas and Eckstein [1988] give a complete theoretical development of the auction algorithm. This algorithm critically depends on the ideas of ϵ-complementary slackness and adaptive scaling. A recent version of Bertsekas' auction code was completed in June 1988 and has been placed in the public domain. Barr and Christiansen [1989] have experimented with a parallel version of this algorithm written in C++ on the Sequent Symmetry S81, and Phillips and Zenios [1989] have experimented with this algorithm on the Connection Machine. Perry [1988] experimented with a parallel version of the auction algorithm on both the Alliant FX/8 and the Sequent Symmetry S81.

Hung and Rom [1980] presented a shortest augmenting path method which had both a polynomial bound and good computational results. Other variants of the shortest augmenting path method have been presented by Glover, Glover and Klingman [1986] and

Jonker and Volgenant [1987]. An extensive computational study with the shortest augmenting path method may be found in Derigs [1985].

Recently, scaling based algorithms have been presented for the assignment problem (see Gabow [1985] and Orlin and Ahuja [1988]). These algorithms are derivatives of the Hungarian method and the auction algorithm, respectively, with the added feature of data scaling. Bertsekas is using a similar idea in his June 1988 auction code.

There are three ways to analyze the performance of an algorithm: worst-case analysis, average case analysis, and empirical analysis. The worst-case analysis results for the assignment problem are presented in Ahuja, Magnanti, and Orlin [1989]. The objective of our study is to present an empirical analysis of two of the fastest serial codes. These codes were obtained directly from the authors, and they represent the current best software implementation of the top competing algorithms. They were run on three different machines (IBM 3081D, Sequent Symmetry S81, and VAX 750) to allow for an analysis with respect to differences in machine architecture and compiler. The shortest augmenting path code was then parallelized and the speedup achieved on a shared memory multiprocessor was reported.

1. SEQUENTIAL CODES

Five algorithms for solving dense assignment problems have been implemented and computationally compared by various researchers. We are not aware of any computational studies involving the dual algorithms. Derigs [1985] shows that a shortest augmenting path implementation is superior to a Hungarian implementation. This has been confirmed by Jonker and Volgenant [1987] and by the authors in a comparison with the codes of Jonker and Volgenant [1987] and Rardin [1986]. Glover, Glover and Klingman [1977] found that their shortest augmenting path code was superior to the specialized simplex code of Barr, Glover and Klingman [1977]. The authors have confirmed this with a comparison of the Jonker and Volgenant [1987] and Barr, Glover and Klingman [1977] codes. Jonker and Volgenant [1987] also concluded that their dense shortest augmenting path code was superior to the auction code of Bertsekas [1981].

After many studies over a fifteen year period, it is acknowledged that the two best algorithms for dense assignment problems are the auction algorithm and the shortest augmenting path algorithm. One of the best software implementations of the auction algorithm is the code of Bertsekas (Version 1.0, June 1988). Some of the other auction codes are very sensitive to the cost structure and degrade as the cost range becomes larger. These other codes sometimes work very well for small cost ranges and fail for cost ranges as small as [0,1000]. By the use of adaptive scaling, Bertsekas' code works well for both a small cost range and a large cost range. This code scales all cost data by n+1 and solves a sequence of problems with decreasing values of the stopping criterion. All calculations are performed in integer arithmetic. Two new parallel auction codes which do not use adaptive scaling may be found in Kempka, Kennington, and Zaki [1991]. An excellent empirical analysis comparing a parallel version of the auction algorithm with a parallel version of the Jonker–Volgenant code on an Alliant FX/8 may be found in Zaki [1990].

We believe that the best implementation of the shortest augmenting path algorithm for serial machines was developed by Jonker and Volgenant [1987]. Dijkstra's algorithm is used to obtain the shortest augmenting paths and only integer arithmetic is required. The code also incorporates an elaborate pre-processing stage which greatly reduces the total number of times that the Dijkstra algorithm is required. It also uses a clever data structure for updating the dual variables after a shortest augmenting path has been found. The code maintains dual variables and the reduced costs are calculated as required. Both codes are written in standard FORTRAN.

The empirical results of our experiment are presented in Table 1 and Figure 1. All times exclude input and output but include the pre-processing. For all test problems, all cost ranges, and all machines, the shortest augmenting path code dominated the auction code. The auction code had the greatest difficulty when the cost range was the smallest, i.e. [0,100]. When the cost range was at least [0,1000], the auction algorithm was affected very little by the cost range. The shortest augmenting path code was adversely affected by an increasing cost range. The machine type affected the empirical analysis. On the Sequent, the shortest augmenting path code was 4.41 times faster than the auction code, on the IBM it was 3.87 times faster, while on the VAX it was 2.79 times faster. This confirms our belief that the comparative performance of two codes is intimately linked to the hardware, operating system, and compilers used. For our tests the IBM was running CMS and both the Sequent and the VAX were running UNIX™†. Our results contradict the widely held belief that the auction algorithm converges faster for lower cost ranges. Bertsekas' auction code for dense problems was not sensitive to changing the cost range from [0,1000] to [0,100000]. In fact the shortest augmenting path code is much more sensitive to the larger cost ranges than the auction code is. We also observed the well-known phenomenon of the auction algorithm that a significant amount of the computational time is spent attempting to complete the last few assignments. This is in contrast to the shortest augmenting path algorithm that achieves one more assignment with each application of Dijkstra's algorithm. Each application of the shortest path algorithm can be very expensive, but it is guaranteed to result in one more assignment. This feature along with the extensive pre-processing to obtain a good set of partial assignments makes this approach work extremely well.

† UNIX is a trade mark of AT&T Bell Laboratories.

We also developed a modification of the shortest augmenting path code which used a Dijkstra two-tree shortest path algorithm (see Helgason, Kennington and Stewart [1988]), but that system was not competitive with the original shortest augmenting path implementation. At the termination of the classical Dijkstra shortest path algorithm, all the information required to update the duals is available and the dual update can be executed very efficiently. The two-tree Dijkstra method can obtain the shortest augmenting path faster than the classical Dijkstra shortest path method; however, additional work is required to discover which duals must be changed and by what amount. The overhead required for the dual variable updates exceeded the potential benefits of the two-tree Dijkstra method for finding the shortest augmenting path.

2. A PARALLEL SHORTEST AUGMENTING PATH CODE

This section contains a description of the algorithm followed by an empirical analysis which identifies the computationally expensive steps of the software implementation of this method. By using prescheduled data partitioning, the operations of these computationally expensive steps were allocated among multiple processors and speedup was measured for computational experiments using up to ten processors.

The algorithm of Jonker and Volgenant [1987] may be divided into three procedures as follows:

(i) column reduction,

(ii) augmenting row reduction, and

(iii) augmentation using a shortest path procedure.

Procedures (i) and (ii) are heuristics which provide an advanced starting solution for procedure (iii) which is an exact method. The details of the algorithms are given below.

THE SHORTEST AUGMENTING PATH ALGORITHM (SAP)

Input:

1. The problem size, n.
2. The nxn cost matrix, c(i, j).

Output:

1. x[i] = j implies that man i is assigned to job j.
2. y[j] = i implies that job j is assigned to man i.
3. v[j] denotes the dual variable associated with job j.

```
begin
    call procedure COLUMN REDUCTION
    call procedure AUGMENTING ROW REDUCTION
    call procedure BUILD A TREE
end
```

```
procedure COLUMN REDUCTION

  begin
1.    $x[i] \leftarrow 0$, $i = 1, \ldots, n$;
2.    for $j = 1, \ldots, n$
3.        $\mu \leftarrow \min \{c[i, j] : i = 1, \ldots, n\}$;
4.        $v[j] \leftarrow \mu$ and let $i^* \in \{i: c[i, j] = \mu\}$;
5.        if $x[i^*] = 0$, then $x[i^*] \leftarrow j$, $y[j] \leftarrow i^*$;
6.    end for
7.    for $i = 1, \ldots, n$
8.        if $x[i] \neq 0$, then
9.            $\mu \leftarrow \min \{c[i, j] - v[j]: j = 1, \ldots, n$ and $j \neq x[i]\}$;
10.           $v[x[i]] \leftarrow v[x[i]] - \mu$;
11.       end if
12.   end for
  end
```

```
procedure AUGMENTING ROW REDUCTION
  begin
13.   $l \leftarrow 0, t \leftarrow 0$;
14.   for i=1, ..., n
15.       if $x[i] = 0$, then $l \leftarrow l+1, f[l] \leftarrow i$;
16.   end for
17.   if $l = 0$, then terminate with an optimum;
18.   $m \leftarrow l, k \leftarrow 1, l \leftarrow 0$;
19.   $i \leftarrow f[k], k \leftarrow k+1$;
20.   $u_1 \leftarrow \min\{c[i, j] - v[j]: j=1, ..., n\}$, let $j_1 \in \{j: c[i, j] - v[j] = u_1\}$,
      $u_2 \leftarrow \min\{c[i, j] - v[j]: j=1, ..., n \text{ and } j \neq j_1\}$, let $j_2 \in \{j: c[i, j] - v[j] = u_2$
      $\text{and } j \neq j_1\}$;
21.   $i_1 \leftarrow y[j_1]$;
22.   if $u_1 < u_2$, then $v[j_1] \leftarrow v[j_1] + u_1 - u_2$;
23.   else if $i_1 = 0$, then go to 26, else $j_1 \leftarrow j_2, i_1 \leftarrow y[j_1]$;
24.   if $i_1 = 0$, then go to 26;
25.   if $u_1 < u_2$, then $k \leftarrow k-1, f[k] \leftarrow i_1$; else $l \leftarrow l+1, f[l] \leftarrow i_1$;
26.   $x[i] \leftarrow j_1, y[j_1] \leftarrow i$;
27.   if $k \leq m$ go to 19;
28.   $t \leftarrow t+1$;
29.   if $l > 0$ and $t < 2$, then go to 18;
  end
```

```
procedure BUILD A TREE
begin
30.  $m \leftarrow 1$
31.  for $l = 1, \ldots, m$
32.      $i^* \leftarrow f[l]$;
33.      READY $\leftarrow \Phi$, TODO $\leftarrow \{1, \ldots, n\}$, RSINK $\leftarrow \{j: y[j] = 0\}$;
34.      $d[j] \leftarrow c[i^*, j] - v[j]$, $pred[j] \leftarrow i^*$, $j=1, \ldots, n$;
35.      $\mu \leftarrow \min\{d[j]: j \in \text{TODO}\}$, SCAN $\leftarrow \{j: d[j] = \mu,\ j \in \text{TODO}\}$, TODO $\leftarrow$
         TODO\SCAN;
36.      if SCAN $\cap$ RSINK $\neq \Phi$, then go to 45;
37.      for all $j_1 \in$ SCAN
38.          $i \leftarrow y[j_1]$, $h \leftarrow c[i, j_1] - v[j_1] - \mu$;
39.          for all $j \in$ TODO
40.              $p \leftarrow c[i, j] - v[j] - h$;
41.              if $p < d[j]$, then $d[j] \leftarrow p$, $pred[j] \leftarrow i$;
42.          end for
43.          READY $\leftarrow$ READY $\cup \{j_1\}$;
44.      end for
45.      go to 35
46.      $v[j] \leftarrow v[j] + d[j] - \mu$,  for all $j \in$ READY;
47.      let $j \in$ SCAN $\cap$ RSINK
48.      $i \leftarrow pred[j]$, $y[j] \leftarrow i$, $k \leftarrow j$, $j \leftarrow x[i]$, $x[i] \leftarrow k$;
49.      if $i \neq i^*$ go to 48;
50.  end for
end
```

Sixty test problems were run and the percentage of computer time allocated to each of these procedures is tabulated in Table 2. All runs were made on a Symmetry S81 from Sequent Computer Systems, Inc. This Symmetry S81 is a multiprocessor system with 32 Mbytes of shared memory and twenty Intel 80386 cpu's. For this study, all codes used only integer arithmetic and did not make use of math co-processors. Note that for these problems, an average of 18% of the time was consumed by the column reduction, 22% was consumed by the augmenting row reduction, and 60% by the tree building activities. Within these three procedures, steps 3, 9, 20, 35, and 39 through 42 are the most time consuming. The percentage of time allocated to these steps is tabulated in Table 3, with over one-half of the computer time attributed to steps 39 - 42.

By using prescheduled data partitioning on steps 3, 9, 20, and 39-42, a parallel SAP code was developed. This software runs in sequential mode until one of the key steps (3, 9, 20 or 39) is reached. Each of these steps is executed in parallel followed by a process synchronization and a return to sequential mode. The objective is to minimize the overhead for parallel processing. The computational times are shown in Table 4 with the corresponding speedups presented in Table 5. As shown in Table 5 under the column entitled 1 cpu, the overhead for parallel processing is approximately 20%. This means that the speedup is limited to at most 5 and the actual speedups using ten processors ranged from a low of 3.27 to a high of 4.32. As expected, the speedups improved as the problem size increased (see Figure 2). The speedups were also slightly better for the fifteen problems having the smallest cost range. This is due to the fact that the pre-processing heuristics are more effective on these problems as shown in Table 2 and the pre-processing steps require less overhead for parallel processing than does the tree building procedure. We also parallelized step 35; however, the synchronization required for updating the SCAN and TODO lists exceeded the benefit of the parallelization.

For the problem sizes and cost ranges analyzed, the times in Table 4 are the best times that we have seen on the Symmetry S81. The parallel shortest augmenting path code is a powerful tool that can easily solve all 1,000,000 arc dense assignment problems in less than sixteen seconds using six processors and less than twelve seconds using ten processors.

Experiments with a parallel version of the auction algorithm on a Sequent Symmetry S81 may be found in Barr and Christiansen [1989]. They used their parallel assignment code to solve a 1,000,000 arc dense assignment problem with a cost range of [1,1000]. Their best time on a Sequent Symmetry S81 using six processors exceeded five minutes. Performance characteristics of the Jacobi and Gauss-Seidel versions of Bertsekas' auction algorithm on an Alliant FX/8 may be found in Kempa, Kennington, and Zaki [1990]. They found that the Gauss-Seidel version was generally more efficient in the scalar environment, but the Jacobi version was generally more efficient in the parallel environment. The observed speedup due to vectorization was higher than that due to concurrency for both versions. The performance characteristics of several software implementations of this algorithm for the Connection Machine CM2 may be found in Wein and Zenios [1990a, 1990b]. Their experimentation indicated that the auction algorithm was not well suited for the architecture of the CM2. Bertsekas and Castanon [1989] compare a variety of synchronous and asynchronous implementations of the auction algorithm on the Encore Multimax. They found that their asynchronous implementations were superior to their synchronous systems.

3. SUMMARY AND CONCLUSIONS

The empirical analysis presented in this study indicates that for dense assignment problems having a size up to 800x800, the shortest augmenting path software is faster than the auction algorithm software. This conclusion was based on test runs with sixteen randomly generated test problems with four different cost ranges and run on three different serial machines. Contrary to the widely held belief that the auction algorithm performs worse as the cost range increases, we found this not to be the case. We believe that Bertsekas' Version 1.0, June 1988 implementation has eliminated this difficulty via the use of adaptive scaling. We did observe the difficulty with the "end game" in which an inordinate amount of time is required to complete the last few assignments. The shortest augmenting path method has the attractive feature that each time a shortest path is calculated, one new assignment is made. We found that the shortest augmenting path code was adversely affected by an increasing cost range. As the cost range increases, larger trees must be developed by Dijkstra's algorithm to obtain the shortest path from an unassigned man to an unassigned job.

By using the technique of prescheduled data partitioning, we parallelized the shortest augmenting path code of Jonker and Volgenant [1987] for the Sequent Symmetry S81. Speedups of three to four were achieved on 1200x1200 dense problems using ten processors. Remarkably, 1,000,000 arc dense assignment problems were solved using this parallel code in less than twelve seconds (wall clock time). Even though this code was developed for a particular multiprocessor system with shared memory, it can be used with any shared memory parallel processing system.

REFERENCES

Ahuja, R., T. Magnanti, and J. Orlin, [1989], "Network Flows," Handbooks in Operations Research and Management Science Volume 1: Optimization, Editors G. Ne[illegible]hauser, A. Rinnooy Kan, and M. Todd, North-Holland, Amsterdam, 211-369.

Akgul, M., [1988], "A Sequential Dual Simplex Algorithm for the Linear Assignment Problem," Operations Research Letters, 7, 155-158.

Balinski, M., [1985], "Signature Methods for the Assignment Problem," Operations Research, 33, 527-536.

Balinski, M., [1986], "A Competitive (Dual) Simplex Method for the Assignment Problem," Mathematical Programming, 34, 125-141.

Barr, R. and M. Christiansen, [1989], "A Parallel Auction Algorithm: A Case Study in the Use of Parallel Object-Oriented Programming," R. Sharada, et al., Impact of Recent Computer Advances on Operations Research, North-Holland Publishing Company, Amsterdam, 23-32.

Barr, R., F. Glover and D. Klingman, [1977], "The Alternating Basis Algorithm for Assignment Problems," Mathematical Programming, 13, 1-13.

Bertsekas, D., [1979], "A Distributed Algorithm for the Assignment Problem," Laboratory for Information and Decision Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139.

Bertsekas, D., [1981], "A New Algorithm for the Assignment Problem," Mathematical Programming, 21, 152-171.

Bertsekas, D., [1988], "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem," Annals of Operations Research, 14, 105-123.

Bertsekas, D., and J. Eckstein, [1988], "Dual Coordinate Step Methods for Linear Network Flow Problems," Mathematical Programming, Series B, 42, 203-243.

Bertsekas, D. and D. Castanon, [1989], "Parallel Synchronous and Asynchronous Implementations of the Auction Algorithm", to appear in Parallel Computing.

Derigs, U., [1985], "The Shortest Augmenting Path Method for Solving Assignment Problems - Motivation and Computational Experience," Annals of Operations Research, 4, 57-102.

Gabow, H., [1985], "Scaling Algorithms for Network Problems," Journal of Computer and System Sciences, 31, 148-168.

Glover, F., R. Glover and D. Klingman, [1986], "Threshold Assignment Algorithm," Mathematical Programming Study, 26, 12-37.

Goldfarb, D., [1985], "Efficient Dual Simplex Algorithms for the Assignment Problem," Mathematical Programming, 33, 187-203.

Helgason, R., J. Kennington and D. Stewart, [1988], "Dijkstra's Two-Tree Shortest Path Algorithm," Technical Report 88-OR-13, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, Texas 75275.

Hung, M., [1983], "A Polynomial Simplex Method for the Assignment Problem," Operations Research, 31, 595-600.

Hung, M. and W. Rom, [1980], "Solving the Assignment Problem by Relaxation," Operations Research, 28, 969-982.

Jonker, R. and T. Volgenant, [1987], "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems," Computing, 38, 325-340.

Kempka, D., J. Kennington, and H. Zaki, [1991], "Performance Characteristics of the Jacobi and the Gauss-Seidel Versions of the Auction Algorithm on the Alliant FX/8," ORSA Journal on Computing, 3, 92-106.

Kennington, J. and R. Helgason, [1980], Algorithms for Network Programming, John Wiley and Sons, New York, NY.

Kuhn, H., [1955], "The Hungarian Method for the Assignment Problem," Naval Research Logistics Quarterly, 2, 83-97.

Kuhn, H., [1956], "Variants of the Hungarian Method for Assignment Problems," Naval Research Logistics Quarterly, 3, 253-258.

McGinnis, L., [1983], "Implementation and Testing of a Primal-Dual Algorithm for the Assignment Problem," Operations Research, 31, 277-291.

Orlin, J. and R. Ahuja, [1988], "New Scaling Algorithms for the Assignment and Minimum Cycle Mean Problems," Technical Report, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139

Perry, E., [1988], "Programming Assignment Algorithms on Parallel and Vector Machines," Technical Report Ford Aerospace Corp., Colorado Springs, CO 80908.

Phillips, C. and S. Zenios, [1989], "Experiences with Large Scale Network Optimization on the Connection Machine," R. Sharada, et al., Impact of Recent Computer Advances on Operations Research, North-Holland Publishing Company, Amsterdam, 169-180.

Rardin, R., [1986], "Private Communication."

Wein, J., and S. Zenios, [1990a], "Massively Parallel Auction Algorithms for the Assignment Problem", In Proceedings of the 3rd Symposium on the Frontiers of Massively Parallel Computation, IEEE Computer Society Press, Los Alamitos, CA 90-99.

Wein, J. and S. Zenios, [1990b], "On the Massively Parallel Solution of the Assignment Problem", Technical Report, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104.

Zaki, H., [1990], "A Comparison of Two Algorithms for the Assignment Problem," Technical Report ORL 90-002, Department of Mechanical and Industrial Engineering, University of Illinois at Urbana-Champaign, Urbana, Il 61801.

Table 1. Comparison of Sequential Algorithms on Dense n x n Assignment Problems (all times are in seconds)

n	cost range	auciion version 1.0 June 1988			shortest augmenting path algorithm SAP		
		IBM 3081D	Symmetry S81	VAX 750	IBM 3081D	Symmetry S81	VAX 750
200	0 - 100	2.36	12.78	42.4	0.67	1.19	5.7
	0 - 1000	2.13	5.19	15.8	0.89	1.56	6.9
	0 - 10000	2.08	5.98	20.8	1.02	1.83	8.2
	0 - 100000	2.14	6.61	22.5	1.70	3.06	12.6
400	0 - 100	16.48	33.23	111.4	2.98	5.03	27.3
	0 - 1000	9.60	13.99	51.6	3.40	5.83	26.4
	0 - 10000	9.55	17.45	62.3	3.50	6.09	38.7
	0 - 100000	10.23	19.54	69.0	3.91	6.90	41.9
600	0 - 100	46.32	228.13	741.6	5.55	9.42	59.7
	0 - 1000	29.58	42.73	158.8	7.30	12.58	75.0
	0 - 10000	27.73	43.08	154.5	8.15	14.26	78.6
	0 - 100000	30.66	52.08	187.8	9.56	16.82	93.6
800	0 - 100	83.41	101.38	364.2	8.43	14.25	79.3
	0 - 1000	42.69	67.33	251.2	11.22	19.22	130.4
	0 - 10000	48.32	73.54	272.4	16.62	28.72	144.0
	0 - 100000	43.43	80.31	288.4	20.23	35.57	182.2
TOTAL		406.71	803.35	2814.7	105.13	182.33	1010.5

Table 2. Total Time Allocated to the Major Routines (all times are the average of five dense n x n assignment problems)

n	cost range	Sequential SAP code (secs.)	COLUMN REDUCTION		AUGMENTING ROW REDUCTION		BUILD A TREE	
			secs.	%	secs.	%	secs.	%
1000	0 - 100	18.37	5.00	27.2	6.65	36.2	6.72	36.6
	0 - 1000	31.45	6.84	21.7	4.44	14.1	20.17	64.1
	0 - 10000	38.49	6.86	17.8	7.39	19.2	24.24	63.0
	0 - 100000	39.43	6.89	17.5	10.20	25.9	22.34	56.7
1100	0 - 100	22.32	6.14	27.5	8.07	36.2	8.12	36.4
	0 - 1000	39.60	8.13	20.5	5.70	14.4	25.80	65.2
	0 - 10000	60.86	8.49	14.0	10.40	17.1	41.97	69.0
	0 - 100000	63.38	8.39	13.2	19.39	30.6	35.60	56.2
1200	0 - 100	26.94	7.18	26.7	10.23	38.0	9.52	35.3
	0 - 1000	47.49	9.65	20.3	6.67	14.0	31.17	65.6
	0 - 10000	60.19	9.91	16.5	10.80	17.9	39.48	65.6
	0 - 100000	60.10	9.96	16.6	11.77	19.6	38.36	63.8
TOTALS		508.62	93.44	18.4	111.71	22.0	303.49	59.6

Table 3. Percentage of Total Time Attributed to the Most Time Consuming Tasks (all times are the average of five dense n x n assignment problems)

n	cost range	Sequential SAP code (secs.)	Step 3		Step 9		Step 20		Step 35		Steps 39–42	
			secs.	%	secs.	%	secs.	%	secs.	%	secs.	%
1000	0 - 100	18.26	4.32	23.7	0.44	2.4	6.65	36.4	1.69	9.3	2.98	16.3
	0 - 1000	31.35	4.32	13.8	2.28	7.3	4.43	14.1	0.41	1.3	19.27	61.5
	0 - 10000	39.00	4.30	11.0	2.36	6.1	7.39	18.9	0.69	1.8	23.78	61.0
	0 - 100000	39.42	4.32	11.0	2.36	6.0	10.21	25.9	0.97	2.5	21.15	53.7
1100	0 - 100	22.40	5.34	23.8	0.61	2.7	8.09	36.1	2.12	9.5	3.61	16.1
	0 - 1000	39.26	5.26	13.4	2.49	6.3	5.70	14.5	0.58	1.5	24.47	62.3
	0 - 10000	59.50	5.27	8.9	2.97	5.0	10.44	17.5	1.07	1.8	39.07	65.7
	0 - 100000	65.74	5.26	8.0	2.88	4.4	19.37	29.5	1.70	2.6	35.81	54.5
1200	0 - 100	26.61	6.29	23.6	0.55	2.1	10.22	38.4	2.78	10.4	3.39	12.7
	0 - 1000	48.08	6.39	13.3	2.86	5.9	6.66	13.9	0.61	1.3	30.57	63.6
	0 - 10000	59.59	6.30	10.6	3.32	5.6	10.80	18.1	0.92	1.5	37.59	63.1
	0 - 100000	59.91	6.32	10.5	3.33	5.6	11.78	19.7	1.23	2.1	36.63	61.1
TOTALS		509.12	63.69	12.5	26.45	5.2	111.74	21.9	14.77	2.9	278.32	54.7

Table 4. The Parallel Shortest Augmenting Path Code (all times are the average in seconds for five dense n x n assignment problems)

n	cost range	Sequential SAP code (secs.)	cpu's used with parallel shortest augmenting path code									
			1	2	3	4	5	6	7	8	9	10
1000	0 - 100	18.20	23.58	12.38	9.77	8.07	6.52	6.23	5.41	5.15	4.88	4.81
	0 - 1000	31.56	39.26	24.06	17.46	14.42	12.93	11.84	10.28	9.70	9.30	8.93
	0 - 10000	40.00	48.71	29.48	21.29	18.03	15.13	13.30	12.99	12.03	11.50	11.00
	0 - 100000	39.23	50.10	29.97	22.59	18.79	16.79	15.43	13.75	13.64	13.10	11.98
1100	0 - 100	22.49	28.81	15.23	11.02	9.71	8.30	7.18	6.68	6.41	5.75	5.54
	0 - 1000	40.41	49.84	30.32	21.52	18.21	15.37	13.97	12.64	11.92	11.59	10.67
	0 - 10000	60.54	73.59	42.45	33.29	27.20	22.57	20.84	19.07	17.92	17.55	16.69
	0 - 100000	63.45	80.45	47.94	34.92	29.59	26.76	23.78	21.18	20.55	19.51	19.21
1200	0 - 100	26.64	34.54	17.85	12.92	10.68	8.92	7.94	7.64	6.97	6.50	6.17
	0 - 1000	47.65	59.64	33.17	25.97	20.94	17.40	16.67	14.74	13.51	12.69	14.18
	0 - 10000	59.36	74.68	42.74	32.22	26.69	22.35	20.54	18.59	17.28	15.97	16.26
	0 - 100000	59.86	74.13	44.04	32.72	28.05	24.42	21.10	20.54	18.92	17.84	17.47
TOTALS		509.39	637.33	369.63	275.69	230.38	197.46	178.82	163.51	154.00	146.18	142.91

Table 5. Speedup for Dense n x n Assignment Problems

n	cost range	Sequential SAP code	cpu's used with the parallel shortest augmenting path code									
			1	2	3	4	5	6	7	8	9	10
1000	0 - 100	1.00	0.77	1.47	1.86	2.26	2.79	2.92	3.36	3.54	3.73	3.78
	0 - 1000	1.00	0.80	1.31	1.81	2.19	2.44	2.67	3.07	3.25	3.39	3.53
	0 - 10000	1.00	0.82	1.36	1.88	2.22	2.64	3.01	3.08	3.33	3.48	3.63
	0 - 100000	1.00	0.78	1.31	1.74	2.09	2.34	2.54	2.85	2.88	3.00	3.27
1100	0 - 100	1.00	0.78	1.48	2.04	2.32	2.71	3.13	3.37	3.51	3.91	4.06
	0 - 1000	1.00	0.81	1.33	1.88	2.22	2.63	2.89	3.20	3.39	3.49	3.79
	0 - 10000	1.00	0.82	1.43	1.82	2.23	2.68	2.91	3.17	3.38	3.45	3.63
	0 - 100000	1.00	0.79	1.32	1.82	2.14	2.37	2.67	3.00	3.09	3.25	3.29
1200	0 - 100	1.00	0.77	1.49	2.06	2.49	2.99	3.35	3.49	3.82	4.10	4.32
	0 - 1000	1.00	0.80	1.44	1.83	2.28	2.74	2.86	3.23	3.53	3.76	3.36
	0 - 10000	1.00	0.79	1.39	1.84	2.22	2.66	2.89	3.19	3.44	3.72	3.65
	0 - 100000	1.00	0.81	1.36	1.83	2.13	2.45	2.84	2.91	3.16	3.35	3.43
	TOTALS	1.00	0.80	1.38	1.85	2.21	2.58	2.85	3.12	3.31	3.48	3.56

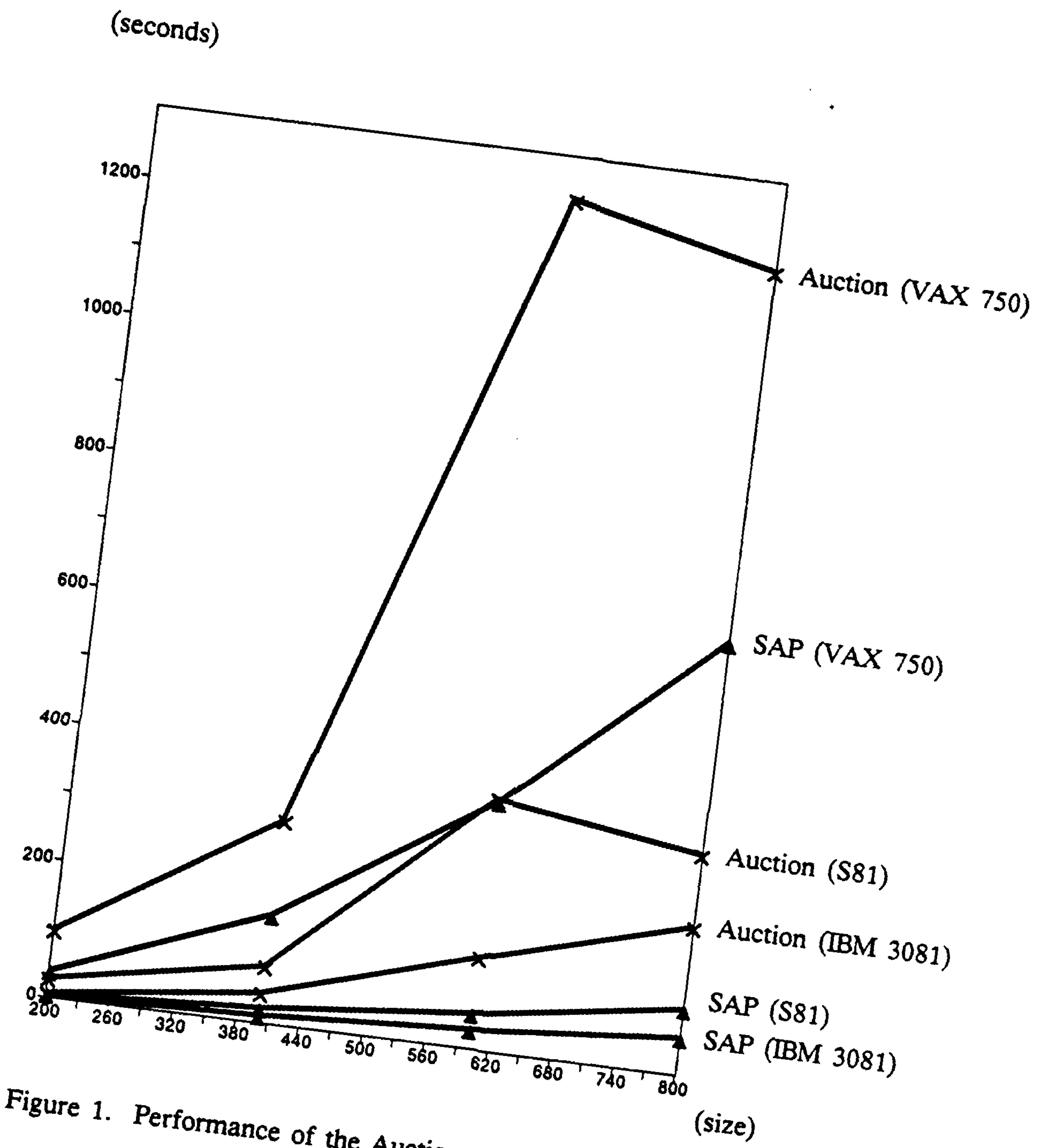

Figure 1. Performance of the Auction and SAP Codes on Different Machines

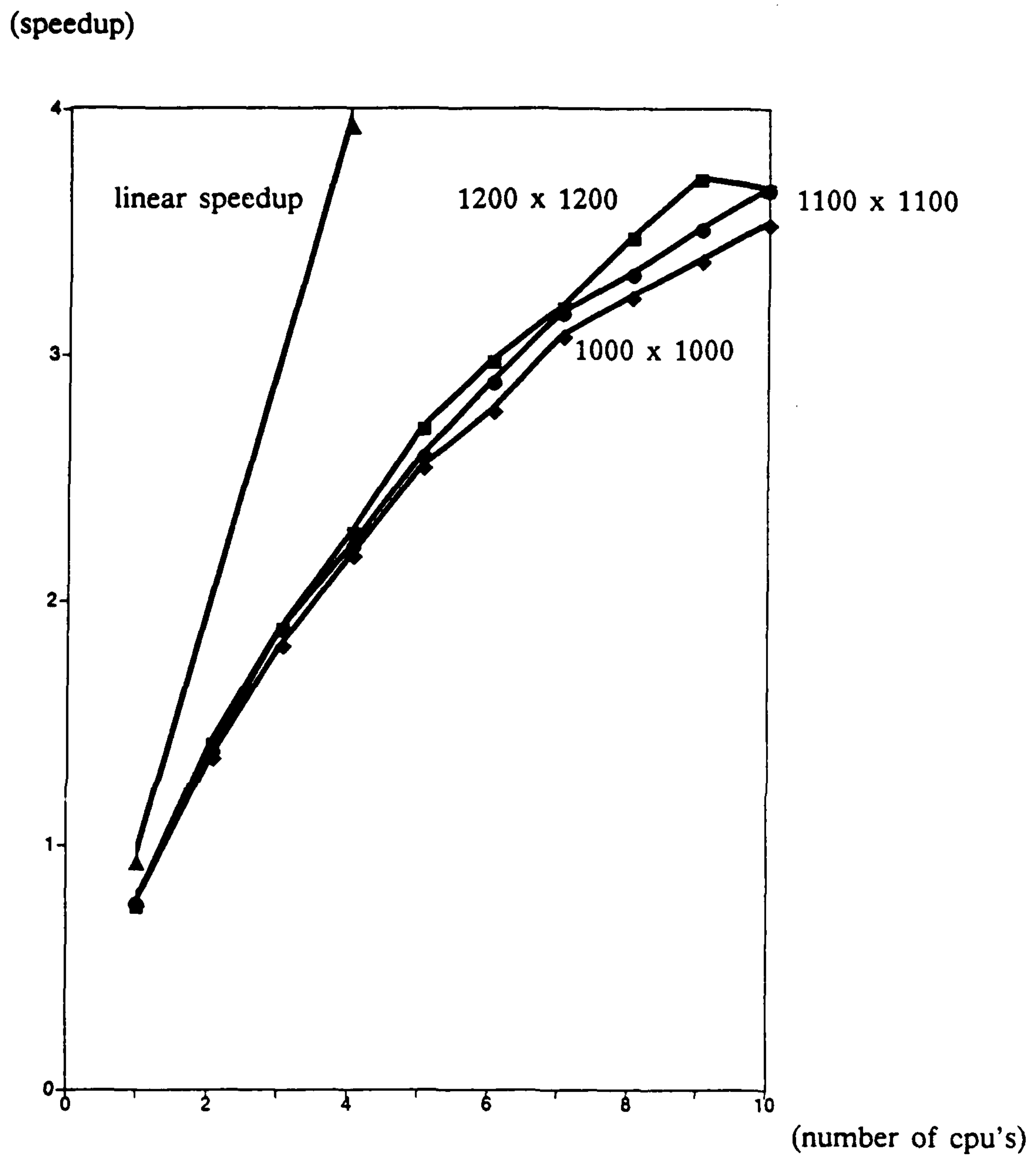

Figure 2. Speedup Performance of the Parallel SAP Code On n x n Dense Assignment Problems

Technical Report 91-CSE-1

THE SINGLY CONSTRAINED ASSIGNMENT PROBLEM

by

Jeffery L. Kennington
Farin Mohammadi

Department of Computer Science and Engineering
School of Engineering and Applied Science
Southern Methodist University
Dallas, Texas 75275-0122
(214)-692-3278

revised December 1991

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

This manuscript presents a new heuristic algorithm to find near optimal integer solutions for the singly constrained assignment problem. The method is based on Lagrangean duality theory and involves solving a series of pure assignment problems. The software implementation of this heuristic successfully solved problems having one-half million binary variables (assignment arcs) in less than fifteen minutes of wall clock time on a Sequent Symmetry S81 using a single processor. In no case did our heuristic fail to obtain a feasible integer solution guaranteed to be within 10% of an optimum. In computational comparisons with MPSX and OSL on an IBM 3081D, the specialized software was from one hundred to one thousand times faster. Our software proved to be very robust as well as fast The robustness is due to an elaborate scheme used to update the Lagrangean multipliers and the speed is due to the fine code used to solve the pure assignment problems. We also present a modification of the algorithm for the case in which the number of jobs exceeds the number of men along with an empirical analysis of the modified software.

ACKNOWLEDGMENT

This research was supported in part by the Air Force Office of Scientific Research under Contract Number AFSOR 91-0028, and the Office of Naval Research under Contract Number N00014-87-K-0223.

I. INTRODUCTION

The singly constrained assignment problem is to determine a least cost assignment of n men to n jobs such that a single additional constraint is satisfied. This model is a special case of a binary linear program and may be stated mathematically as follows:

$$\text{minimize} \quad \sum_{(i,j) \in E} c_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{j : (i,j) \in E} x_{ij} = 1 \ , \quad i = 1, \dots, n \tag{2}$$

$$\sum_{i : (i,j) \in E} x_{ij} = 1 \ , \quad j = 1, \dots, n \tag{3}$$

$$x_{ij} \in \{0, 1\} \ , \quad \text{all } (i,j) \in E \tag{4}$$

$$\sum_{(i,j) \in E} a_{ij} x_{ij} \leq b \tag{5}$$

where c_{ij} denotes the cost for assigning man i to job j, a_{ij} denotes the coefficient of x_{ij} in the side constraint, b denotes the right-hand-side of the side constraint, E is the set of (man, job) pairs corresponding to eligible assignments, and $x_{ij} = 1$ implies that man i is assigned to job j. In order to simplify the notation we let a denote the vector corresponding to the coefficients in (5), x denote the vector corresponding to the binary decision variables, c denote the vector of costs, and T= { x : (2), (3), and (4) }. Then the singly constrained assignment problem can be stated as $P_1 = \min \{cx : x \in T \text{ and } ax \leq b\}$, and it is well-known that P_1 is NP-complete.

The singly constrained assignment was first used by Brans, Leclercq, and Hansen [1973] to model the core management of a pressurized water reactor. The problem is given two sets of fresh and exposed assemblies determine the location pattern of these assemblies which maximizes the reactivity of the core under a constraint on power-distribution form factor. After linearization, Brans, Leclercq, and Hansen reduce the problem to a sequence of singly constrained assignment problems and propose an implicit enumeration routine to solve these problems. Our work on P_1 was motivated by models which had been developed by analysts at the Navy Personnel Research and Development Center in San Diego. These models involve the optimal assignment of men to jobs under a budget constraint related to relocation cost.

The first specialized algorithm for P_1 was presented by Gupta and Sharma [1981]. Their method was a straight forward enumeration scheme and they present no computational results. Aggarwal [1985] presents an improved algorithm for P_1 which combines Lagrangean-relaxation with the enumeration algorithm of Gupta and Sharma [1981] to obtain an optimal solution. No computational results for this method is presented. Mazzola and Neebe [1985] develop a branch-and-bound algorithm for the constrained assignment problem. To generate solutions at each node of the branch-and-bound tree they developed a method that combines a restricted basis pivoting rule followed by a subgradient routine. Their empirical evaluation of the heuristic and the branch-and-bound algorithm indicates that both procedures are satisfactory for dense assignment problems of size up to 100x100. Bryson [1991] presents an algorithm based on the parametric programming procedure of Gass and Saaty [1955]. The largest problem they solved had fewer than

2000 edges and did not exploit the network structure of this model. This is in contrast to our empirical investigation in which the small problems have over a quarter of a million edges. Ball, Derigs, Hilbrand, and Metz [1990] present an algorithm which will solve the special case of P_1 in which $a_{ij} \in \{0,1\}$ for all (i,j).

The work by Klingman and Russell [1975] and Barr, Farhangian, and Kennington [1986] is for the continuous version of P_1 rather than the binary version. That is, the above work would be applicable for the model in which (4) is replaced with the nonnegativity constraint $x_{ij} \geq 0$, all $(i,j) \in E$.

Klingman and Russell [1978] developed a simplex based method for the transportation problem with a single side constraint and Glover, Karney, Klingman, and Russell [1978] developed a simplex based method for the transshipment problem with a single side constraint. Authors of both papers state that codes based on their procedures are significantly faster than the LP code APEX–III and they both obtain an integer solution for the problem with an inequality side constraint by pivoting into the basis the slack variable associated with the side constraint. This yields a triangular basis which automatically produces an integer solution. Empirically these integer solutions were found to be within 1% of optimality.

Since (1)–(5) is a binary linear program, all the literature on integer programming applies (see Everett [1963], Geoffrion [1969, 1974], Salkin [1974], Shapiro [1971, 1979], Parker and Rardin [1988], Nemhauser and Wolsey [1988]). In practice most integer programming models are either solved as a linear program and the solutions are rounded using some heuristic or branch–and–bound is used in an attempt to obtain a solution within a prespecified tolerance.

The objective of this study is to present a new algorithm for the singly constrained assignment problem. The algorithm is for the problem having an inequality side constraint. We also show how this algorithm can be used to solve problems in which (5) is an equality and problems in which (3) is replaced with

$$\sum_{i\,:\,(i,j)\,\in\,E} x_{ij} \quad \leq \quad 1 \;,\; j = 1, \ldots, m. \qquad (6)$$

The algorithm uses a Lagrangean relaxation and solves a series of assignment problems. Empirical results demonstrate the superiority of this approach over competing software. Problems having one-half million arcs were solved in less than fifteen minutes on a Sequent Symmetry S81 using a single processor.

II. THE ALGORITHM

In this section we present a heuristic algorithm for the singly constrained assignment problem, $P_1 = \min \{ cx : x \in T,\ ax \leq b \}$. Let $P_2 = \min \{ cx : x \in T \}$ be a feasible region relaxation of P_1, and let $P_3 = \min \{ ax-b : x \in T \}$. By dualizing the side constraint one obtains a Lagrangean relaxation of P_1 given by $P(\beta) = \min \{ cx + \beta(ax-b) : x \in T \}$ where β is the Lagrangean multiplier. Let $v[P]$ denote the optimal objective function value for any problem P, then a Lagrangean dual for P_1 is $Dl_1 = \max \{ v[P(\beta)] : \beta \geq 0 \}$.

We attempt to solve the Lagrangean dual, Dl_1, by solving the problems P_2, P_3 and a series of $P(\beta)$ for different values of β. P_2 is solved to obtain the initial lower bound, lb, and to determine if the side constraint is redundant. The solution to P_3 either establishes that P_1 has no feasible solution or provides an initial upper bound, ub. Solving the Lagrangean relaxation, $P(\beta)$, always provides a lower bound and if the optimal solution, x_β, is feasible for P_1, then cx_β is an upper bound.

It is well known that $v[P(\beta)]$ is a piece-wise linear concave function over R^+. Let x_β denote an optimum for $P(\beta)$ at any point β. Let β^* denote an optimum for Dl_1. The optimum for Dl_1 may be unique as illustrated in Figure 1 or may have an infinite number of solutions as illustrated in Figure 2. For the case illustrated in Figure 1, for all $\beta > \beta^*$, $ax_\beta < b$ and x_β is feasible for P_1, and for all $\beta < \beta^*$, $ax_\beta > b$ and x_β is not feasible for P_1. For the case illustrated in Figure 2, for all $\beta > \beta^*$, either $ax_\beta < b$ and x_β is feasible for P_1 or $ax_\beta = b$ and x_β is an optimum for P_1. Similarly for all $\beta < \beta^*$, either $ax_\beta > b$ and x_β is not feasible for P_1 or $ax_\beta = b$ and x_β is an optimum for P_1.

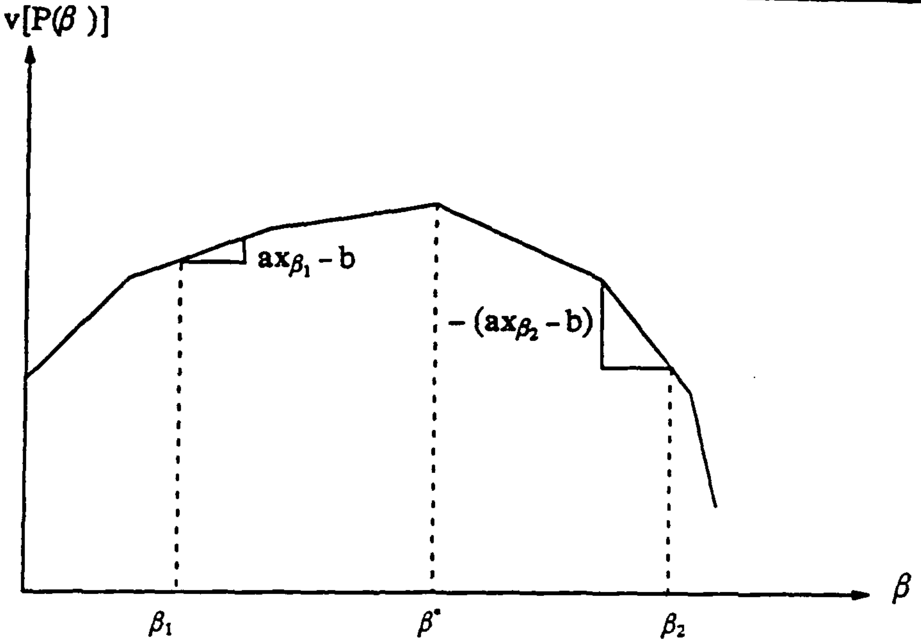

Figure 1. Unique optimal solution for DI_1

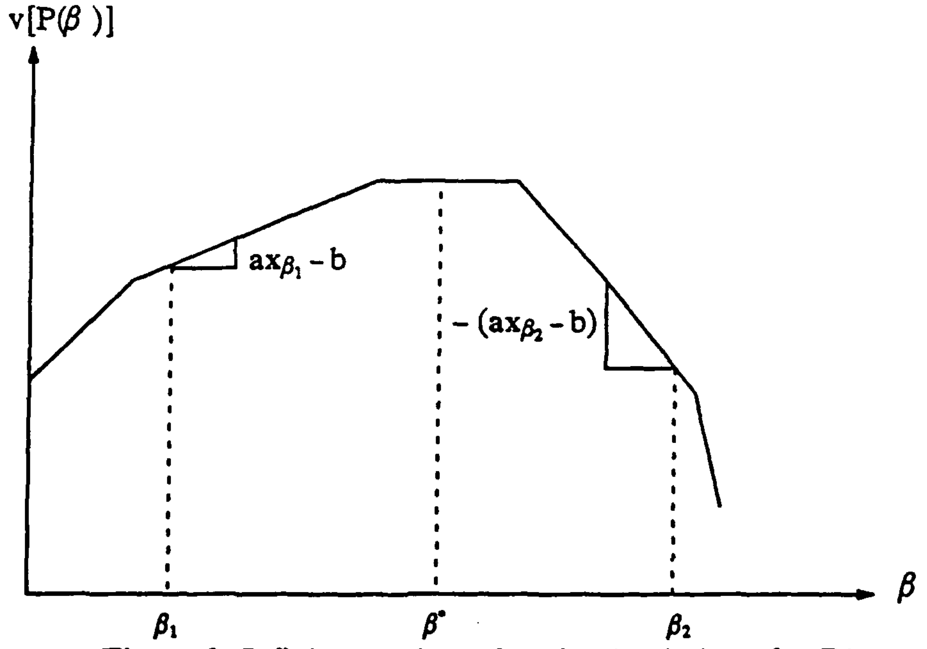

Figure 2. Infinite number of optimal solutions for DI_1

Let u and v denote upper and lower bounds, respectively on β^*. Then a bisection search can be used to obtain β^*. Since obtaining each value of $v[P(\beta)]$ for a given β requires solving an assignment problem, we attempt to obtain a small interval of uncertainty $[u,v]$ prior to initiating the bisection search. It is well-known that if the duality gap is zero, then $v[DI_1]=v[P_1]$. That is,

$$cx_{\beta^*} + \beta^*(ax_{\beta^*} - b) = v[P_1] \text{ , or} \tag{7}$$

$$\beta^* = (v[P_1] - cx_{\beta^*})/(ax_{\beta^*} - b). \tag{8}$$

Prior to using the bisection search, we obtain estimates for β^* using (8) with $v[P_1]$ replaced by $(lb+ub)/2$, cx_{β^*} replaced by ub, and ax_{β^*} replaced by ax_3. That is, initial values of β are given by:

$$\beta = (ub - lb)/(2(ax_3 - b)). \tag{9}$$

The basic strategy of a heuristic algorithm for P_1 follows:

```
step 1.  find an initial lower bound,
step 2.  find an initial upper bound,
while  $ax_\beta < b$ and stopping criteria not satisfied repeat steps 3 and 4,
step 3.  use (8) to estimate $\beta$ then solve $P(\beta)$,
step 4.  update $u$ and $ub$ and if possible $v$ and $lb$,
while stopping criteria is not satisfied repeat steps 5-7,
step 5.  let $\beta = (u+v)/2$ and solve $P(\beta)$,
step 6.  if $ax_\beta < b$ update $u$, $ub$, and if possible $lb$,
step 7.  if $ax_\beta > b$ update $v$ and if possible $ub$ and $lb$.
```

The algorithm terminates if one of the following conditions is satisfied:

(i) $ax_\beta=b$, since by strong Lagrangean duality x_β is an optimum.

(ii) $ub-lb \leq .1\ lb$, since for Navy Personnel Research and Development Center models, an assignment that is guaranteed to be within 10% of an optimum is considered to be acceptable.

(iii) *iter* = maxiter, since we cannot guarantee that criterion (i) or (ii) will ever be satisfied, we also stop after solving a prespecified number of assignment problems.

The ASSIGN+1 algorithm for P_1 is described below:

Input:

1. The cost vector, c.
2. The side constraint, vector a and constant b.
3. The set of (man, job) pairs corresponding to eligible assignments, E.
4. The maximum number of iterations permitted before termination, maxiter.

Output:

1. The solution vector, y.
2. A lower bound for P_1, *lb*.
3. The objective value corresponding to y, *ub*.
4. The termination status. If P_1 has no feasible solution, then *status* = infeasible; if an optimal solution was obtained then *status*=optimal; otherwise, *status* = feasible.

```
algorithm ASSIGN+1:
begin
    $iter$:=0, $v$:=0;
    FIND INITIAL LOWER BOUND;
    FIND INITIAL UPPER BOUND;
    while $\gamma>0$ and $iter \leq$ maxiter do FIND INITIAL U;
    while $v=0$, $\gamma<0$, $|ub-lb|>.1|lb|$, and $iter \leq$ maxiter do REDUCE U;
    while $|ub-lb|>.1|lb|$ and $iter \leq$ maxiter do BISECTION;
    if $u>\beta$, then $\beta := u$ and let $x_\beta$ be an optimum for $P(\beta)$;
    $y := x_\beta$;
end;

procedure FIND INITIAL LOWER BOUND
begin
    let $x_2$ be an optimum for $P_2$, $iter := iter+1$, $lb := cx_2$;
    if $ax_2 \leq b$, then $status$:= optimal, $y := x_2$, $ub := cy$, stop;
end;

procedure FIND INITIAL UPPER BOUND
begin
    let $x_3$ be an optimum for $P_3$, $iter := iter+1$, $\delta := ax_3-b$;
    if $\delta>0$, then $status$:= infeasible, stop;
    else $status$:= feasible, $ub := cx_3$, $\gamma := -1$, $\beta := (lb-ub)/(2\delta)$;
end;
```

```
Procedure FIND INITIAL U
begin
    let $x_\beta$ be an optimum for $P(\beta)$, $iter := iter+1$;
    $\gamma := ax_\beta - b$, $lb := \max\{lb, v[P(\beta)]\}$;
    if $\gamma=0$, then $status :=$ optimal, $y := x_\beta$, $ub := cy$, stop;
    if $\gamma<0$ then $ub := \min\{ub, cx_\beta\}$, $u := \beta$;
    else $v := \max\{v,\beta\}$ , $\beta := 2\beta$;
end;

procedure REDUCE U
begin
    if $\beta=(lb-ub)/(2\delta)$, then $\beta := \beta/2$;
    else $\beta := (lb-ub)/(2\delta)$;
    let $x_\beta$ be an optimum for $P(\beta)$, $iter := iter+1$;
    $\gamma := ax_\beta - b$, $lb := \max\{lb, v[P(\beta)]\}$;
    if $\gamma=0$, then $status :=$ optimal, $y := x_\beta$, $ub := cy$, stop;
    if $\gamma<0$, then $ub := \min\{ub, cx_\beta\}$, $u := \min\{u,\beta\}$;
    else $v := \beta$;
end;
```

```
procedure BISECTION
begin
    $\beta$:= $(u+v)/2$;
    let $\mathbf{x}_\beta$ be an optimum for $P(\beta)$, iter:= iter+1;
    $\gamma$:= $\mathbf{a}\mathbf{x}_\beta - b$, lb:= $\max\{lb, v[P(\beta)]\}$;
    if $\gamma=0$, then status:= optimal, y:= $\mathbf{x}_\beta$, ub:= cy, and stop;
    if $\gamma<0$, then ub:= $\min\{ub, c\mathbf{x}_\beta\}$, and u:= $\min\{u,\beta\}$;
    else v:= $\max\{v,\beta\}$;
end;
```

It is well-known that the main difficulty with the Lagrangean approach is the selection of the sequence of multipliers, β, so that software implementations using these rules are robust. Convergence results can be found in Allen, Helgason, Kennington, and Shetty [1987]. Most of the steps in the above algorithm are related to the elaborate scheme for updating β which was developed through empirical analysis.

III. AN EQUALITY SIDE CONSTRAINT

In this section we show that the ASSIGN+1 algorithm is also applicable to the assignment problem with an equality side constraint, $P'_1 = \{ \min cx : x \in T, ax=b \}$. That is, we show that if P'_1 is a feasible problem, then solving P'_1 is equivalent to solving either P_1 or $P''_1 = \{ \min cx : x \in T, ax \geq b \}$ or P_2.

Proposition 1 Let P'_1 be a feasible problem, and let x_2 be an optimum for P_2. If $ax_2 < b$, then $v[P'_1] = v[P''_1]$.

Proof Let F(P) denote the feasible region for any problem P, x''_1 be an optimum for P''_1 and suppose $v[P'_1] \neq v[P''_1]$. Since $v[P'_1] \neq v[P''_1]$, and $F(P'_1) \subset F(P''_1)$, then $v[P'_1] > v[P''_1]$. Therefore $ax''_1 > b$. Let $x''_1+\lambda_1 d_1$ with $0 \leq \lambda_1 \leq 1$ and $d_1 = x_2 - x''_1$ be the line segment with end points x_2 and x''_1. This line segment can also be written as $x_2+\lambda_2 d_2$ with $0 \leq \lambda_2 \leq 1$ and $d_2 = x''_1 - x_2$. Note that d_2 is a feasible direction for P_2 at the point x_2, since $x_2 \in F(P_2)$ and $x''_1 \in F(P_2)$. Let $D_d g(y)$ denote the directional derivative of the function $g(x)$ in the direction d at the point y. Letting $g(x)$ be the linear function cx, we know that $D_{d_1} g(x''_1) = -D_{d_2} g(x_2)$. Since $ax''_1 > b$ and $ax_2 < b$, there exists a $\bar{\lambda}_1$ such that $\bar{x} = x''_1+\bar{\lambda}_1 d_1$ and $a\bar{x}=b$. Since $\bar{x} \in F(P'_1)$ and $v[P'_1] > v[P''_1]$, $c\bar{x} > cx''_1$. Likewise, $c\bar{x} > cx_2$. Hence $D_{d_1} g(\bar{x}) < 0$ and $D_{d_2} g(\bar{x}) < 0$, a contradiction. Hence $v[P'_1] = v[P''_1]$. ∎

Proposition 2 Let P'_1 be a feasible problem, and let x_2 be an optimum for P_2. If $ax_2 > b$, then $v[P'_1] = v[P_1]$.

Proof Similar to the proof for the Proposition 1.

Proposition 3 Let x_2 be an optimum for P_2. If $ax_2 = b$, then x_2 solves P'_1.

Proof Let F(P) denote the feasible region for any problem P, $F(P'_1) \subset F(P_2)$ and $x_2 \in F(P'_1)$, then x_2 is an optimum for P'_1. ∎

As Propositions 1 and 2 indicate, the equality problem can be solved by solving an inequality problem. The solution to P_2 indicates whether one needs to solve the problem with $\mathbf{ax} \leq b$ or $\mathbf{ax} \geq b$. Since finding a set of assignments for which $ax = b$ may not always be possible and since from our work with the Navy Personnel Research and Development Center we have found that most constraints can be slightly violated and acceptable solutions can still be obtained, we replace $ax-b = 0$ with $|ax-b| \leq \epsilon$. That is, instead of P'_1 we attempt to solve $\min \{ cx : \mathbf{x} \in T \text{ and } |ax-b| \leq \epsilon \}$. For all our work we set ϵ to .01b.

IV. EMPIRICAL ANALYSIS

The algorithm ASSIGN+1 has been implemented in software and empirically analyzed on both a Sequent Symmetry S81 and an IBM 3081D for both inequality and equality side constraints. Both codes are written in FORTRAN and use SEMI (see Kennington and Wang [1990a, 1990b]) to solve the assignment problems. SEMI is an implementation of the shortest augmenting path algorithm for sparse semi-assignment problems and is claimed to be one of the fastest codes available for both assignment and semi-assignment problems.

We developed a test problem generator with the following inputs: (i) the number of men, (ii) the arc density, (iii) the maximum cost, $\bar{c}$, and (iv) the side constraint multiplier, k. Both the costs and the side constraint coefficients are uniformly distributed over the range $[0 - \bar{c}]$. We randomly generate a feasible assignment, $\bar{x}$, and determine $\bar{b}$ for this assignment so that $a\bar{x} = \bar{b}$. The right-hand-side, b, for the side constraint is set to $k\bar{b}$. For the inequality problems and k = 1, we observed that for most problems, the side constraint was redundant and therefore a very easy problem. As k becomes smaller, the feasible region becomes smaller and for sufficiently small k the problem may become infeasible.

The generator was used to generate the eighty-one inequality problems described in Table 1. Under the column entitled "Side Const", k was set to .2, .5, and .9 for the rows entitled "small", "medium", and "large", respectively. It should be noted that the software is very robust as a function of the magnitude of b and it requires very few iterations to satisfy the optimality criteria. Near optimal solutions to integer programs with over one-half million binary variables were routinely obtained in less than fifteen minutes.

Table 2 gives our empirical results with 135 equality problems. Under the column entitled "Side Const", k was set to .2, .5, .9, 1.2, and 1.5 for the rows entitled "very small", "small", "medium", "large", and "very large", respectively. The software is very robust over a wide range of input parameters and performed very well on all these problems.

In contrast with some of our previous experience using subgradient optimization, we never found a problem that caused this software any major difficulty. We attribute the robustness of this software to the elaborate scheme for updating the Lagrangean multiplier which works well for this class of problems. We attribute the speed of this software to the semi-assignment software, SEMI. Of course, the version of SEMI that we used was modified to handle real cost as opposed to integer data required by the version described in Kennington and Wang [1990a].

Tables 3 through 6 present our empirical results comparing the specialized software for the singly constrained assignment problem with both MPSX (see Mathematical Programming System Extended [1979]) and OSL (see Optimization Subroutine Library [1990]). If the ASSIGN+1 software for the equality side constraint terminates due to the maximum number of iterations and no feasible solution has been found, then the current best known Lagrangean multiplier is used to find a solution. In this case the side constraint violation will exceed 1%. This occurred for four of sixteen problems presented in Tables 3–6. In the worst case, the maximum deviation from feasibility was 1.65%. That is, all solutions satisfied the constraint $|ax-b| \leq 0.0165b$.

All the MPSX and OSL runs were made with default parameter settings. A few of the smaller problems were successfully solved, but the times were from two to three orders of magnitude slower than those for the specialized software. In addition we ran a specialized network with side constraint code, NETSIDE (see

Kennington and Whisman [1990]) on an 800x800 test problem in an attempt to solve the linear programming relaxation of this model. Convergence was not achieved after two hours of cpu time on the Sequent Symmetry S81.

As stated we have modified SEMI to handle real cost coefficients as opposed to the integer cost coefficients. Generally this modification is expected to increase the execution time drastically, but in this case, as Table 7 indicates this increase was less than ten percent. Results presented in Table 7 are the average wall clock times for three pure assignment problems, running on a Sequent Symmetry S81 using the floating point accelerator.

Table 1. The assignment problem with an inequality side constraint

Problem Size	Cost/Const Range	Side Const RHS	# of Iter	Time[1] (min)	Solution Guaranteed within % of Opt.
800x800 (256,000 arcs)	0-1000	small	7.33	3.03	95.16
		medium	7.33	2.98	93.28
		large	6.00	2.48	97.88
	0-10000	small	8.33	3.56	92.81
		medium	7.33	3.04	94.10
		large	6.00	2.67	98.26
	0-100000	small	8.33	3.62	92.56
		medium	7.33	3.05	94.14
		large	6.00	2.61	98.32
1000x1000 (400,000 arcs)	0-1000	small	6.00	4.26	95.17
		medium	8.33	6.43	92.39
		large	6.00	4.10	97.34
	0-10000	small	6.00	4.42	93.60
		medium	7.67	5.56	93.37
		large	6.00	4.29	97.21
	0-100000	small	6.00	4.51	94.60
		medium	7.67	5.72	93.60
		large	6.00	4.40	97.33
1200x1200 (576,000 arcs)	0-1000	small	6.67	7.70	94.42
		medium	8.33	10.00	92.35
		large	5.00	5.36	99.06
	0-10000	small	6.67	8.18	93.69
		medium	8.33	10.36	95.14
		large	6.00	7.09	96.87
	0-100000	small	6.67	8.31	93.83
		medium	8.33	10.43	95.06
		large	6.00	6.99	96.98

[1] Times are wall clock time on a Sequent Symmetry S81 using one processor.

Table 2. The assignment problem with an equality side constraint

Problem Size	Cost/Const Range	Side Const RHS	# of Iter	Time[1] (min)	Solution Guaranteed within % of Opt.
800x800 (256,000 arcs)	0-1000	very large	8.67	4.06	99.15
		large	10.67	4.94	99.64
		medium	8.67	4.08	99.80
		small	8.67	4.14	99.33
		very small	7.67	3.45	99.44
	0-10000	very large	8.67	4.14	99.39
		large	10.67	4.93	99.76
		medium	9.00	4.34	99.96
		small	9.00	4.26	100.00
		very small	8.67	4.27	99.93
	0-100000	very large	8.67	4.16	99.64
		large	10.67	5.04	99.76
		medium	8.67	4.15	99.91
		small	9.33	4.42	99.95
		very small	10.00	4.53	99.95
1000x1000 (400,000 arcs)	0-1000	very large	9.67	7.82	99.91
		large	10.33	8.32	99.76
		medium	8.00	6.00	99.84
		small	10.33	8.07	99.79
		very small	9.00	7.40	100.00
	0-10000	very large	10.33	8.44	99.61
		large	10.33	8.49	99.84
		medium	7.00	5.36	99.82
		small	10.67	9.06	99.53
		very small	10.67	9.56	100.00
	0-100000	very large	10.33	8.41	99.56
		large	10.67	8.50	99.78
		medium	8.00	6.40	99.77
		small	10.67	8.78	99.44
		very small	10.30	9.04	99.87
1200x1200 (576,000 arcs)	0-1000	very large	10.00	11.93	98.55
		large	9.00	10.39	99.87
		medium	9.33	11.05	99.99
		small	10.33	13.02	99.43
		very small	9.33	11.69	99.74
	0-10000	very large	9.67	10.68	98.40
		large	11.00	14.21	99.82
		medium	8.67	11.13	99.92
		small	10.00	12.00	99.65
		very small	11.00	14.18	99.71
	0-100000	very large	10.33	12.12	98.81
		large	10.67	13.48	99.54
		medium	8.67	10.34	99.92
		small	9.33	11.72	99.76
		very small	11.00	14.45	99.73

[1] Times are wall clock time on a Sequent Symmetry S81 using one processor.

Table 3. Comparisons of specialized algorithms with MPSX and OSL on an IBM 3081D (very small value of b in side constraint)

Problem Name	A100+1		A200+1		A300+1		A400+1	
Problem Characteristics								
Rows	201	201	401	401	601	601	801	801
Columns	4,000	4,000	16,000	16,000	36,000	36,000	64,000	64,000
Side Constraint Type	≤	=	≤	=	≤	=	≤	=
MPSX								
Nodes in BAB Tree	602	1,014	125	77	Too many binary variables	Too many binary variables	Too many binary variables	Too many binary variables
Iter	13,176	12,248	19,325	14,336				
Time (Sec)	815	901	3,343	1,864				
Obj. Value Best Inc.	31,578	33,285	NA	NA				
Status	Optimal	Feasible[1]	Unknown[1]	Unknown[2]	Unknown[3]	Unknown[3]	Unknown[3]	Unknown[3]
OSL								
Nodes in BAB Tree	1,465	2,126	200	200	200	200	Error during input	Error during input
Iter	10,916	36,902	4,827	3,969	10,082	6,405		
Time (Sec) ≥	1,958	564	2,038	943	3,634	3,909		
Obj. Value Best Inc.	31,564	NA	NA	NA	NA	NA	NA	NA
Status	Optimal	Unknown[4]	Unknown[5]	Unknown[5]	Unkown[5]	Unknown[5]	Unknown[6]	Unknown[6]
ASSIGN+1								
Time (Sec)	0.89	1.73	4.13	5.43	13.26	13.24	26.74	43.22
Obj. Value	34,359	30,830	33,361	32,024	30,550	30,550	32,026	31,439
Dev. from Feasibility	0%	1.57%	0%	0.45%	0%	0.98%	0%	0.28%
Max. Dev. from Opt.	8.81%	0%	4.58%	0.46%	1.10%	1.10%	1.59%	0%

[1] Terminated due to node table overflow after obtaining an incumbent but prior to obtaining an optimal solution.
[2] Manually stopped after 30 minutes of CPU time prior to obtaining the first incumbent.
[3] MPSX/370 restricts the number of integer variables to maximum of 32,767.
[4] Terminated due to basis file overflow.
[5] Terminated due to branch and bound node table overflow (node table set to 200 problems).
[6] Input error due to file size (file has 193,481 lines).

Table 4. Comparisons of specialized algorithms with MPSX and OSL on an IBM 3081D (small value of b in side constraint)

Problem Name	A100+1		A200+1		A300+1		A400+1	
Problem Characteristics								
Rows	201	201	401	401	601	601	801	801
Columns	4,000	4,000	16,000	16,000	36,000	36,000	64,000	64,000
Side Constraint Type	≤	=	≤	=	≤	=	≤	=
MPSX								
Nodes in BAB Tree	1,009	1,009	728	915	Too many binary variables	Too many binary variables	Too many binary variables	Too many binary variables
Iter	6,332	4,082	16,324	19,153				
Time (Sec)	320	308	1,789	2,212				
Obj. Value Best Inc.	31,578	33,285	NA	NA				
Status	Unknown[2]	Unknown[1]	Unknown[2]	Unknown[2]	Unknown[3]	Unknown[3]	Unknown[3]	Unknown[3]
OSL								
Nodes in BAB Tree	715	2,054	200	200	200	200	Error during input	Error during input
Iter	5,364	17,261	2,087	2,152	3,097	3,041		
Time (Sec) ≥	907	223	1,187	975	3,338	3,350		
Obj. Value Best Inc.	6,291	NA	NA	NA	NA	NA	NA	NA
Status	Optimal	Unknown[4]	Unknown[5]	Unknown[5]	Unkown[5]	Unknown[5]	Unknown[6]	Unknown[6]
ASSIGN+1								
Time (Sec)	1.28	1.15	4.18	4.13	13.33	18.29	18.26	42.70
Obj. Value	6,376	6,624	5,564	5,564	6,189	5,923	6,288	6,085
Dev. from Feasibility	0%	0.03%	0%	0.93%	0%	0.74%	0%	1.06%
Max. Dev. from Opt.	1.70%	0%	0.97%	0.97%	4.89%	0%	2.27%	0%

[1] Terminated due to node table overflow prior to obtaining the first incumbent.
[2] Terminated due to problem file overflow prior to obtaining the first incumbent .
[3] MPSX/370 restricts the number of integer variables to maximum of 32,767.
[4] Terminated due to basis file overflow.
[5] Terminated due to branch and bound node table overflow (node table set to 200 problems).
[6] Input error due to file size (file has 193,481 lines).

Table 5. Comparisons of specialized algorithms with MPSX and OSL on an IBM 3081D (large value of b in side constraint)

Problem Name	A100+1		A200+1		A300+1		A400+1	
Problem Characteristics								
Rows	201	201	401	401	601	601	801	801
Columns	4,000	4,000	16,000	16,000	36,000	36,000	64,000	64,000
Side Constraint Type	$\leq$	=	$\leq$	=	$\leq$	=	$\leq$	=
MPSX								
Nodes in BAB Tree	1,018	1,009	43	915	Too	Too	Too	Too
Iter	4,129	7,847	13,911	17,135	many	many	many	many
Time (Sec)	428	470	499	1,538	binary	binary	binary	binary
Obj. Value Best Inc.	3,992	NA	3,763	NA	variables	variables	variables	variables
Status	Feasible[4]	Unknown[2]	Optimal	Unknown[2]	Unknown[3]	Unknown[3]	Unknown[3]	Unknown[3]
OSL								
Nodes in BAB Tree	10	2,584	94	200	200	200	Error	Error
Iter	NA	32,992	2,109	3,104	5,798	9,905	during	during
Time (Sec) $\geq$	NA	4,637	908	1,580	5,440	4,885	input	input
Obj. Value Best Inc.	3,911	NA	3,763	NA	3,744	NA	NA	NA
Status	Feasible[4]	Unknown[5]	Unknown[5]	Unknown[5]	Unkown[5]	Unknown[5]	Unknown[6]	Unknown[6]
ASSIGN+1								
Time (Sec)	0.79	1.73	4.41	4.45	10.53	21.04	23.20	31.19
Obj. Value	3,964	3,863	3,771	3,771	3,833	3,731	4,102	3,952
Dev. from Feasibility	0%	1.65%	0%	0.74%	0%	1.01%	0%	0.45%
Max. Dev. from Opt.	2.01%	0%	0.28%	0.28%	2.78%	0%	4.01%	0.12%

[1] Terminated due to node table overflow prior to obtaining the first incumbent.
[2] Terminated due to problem file overflow prior to obtaining the first incumbent .
[3] MPSX/370 restricts the number of integer variables to maximum of 32,767.
[4] Terminated due to OSL data overflow.
[5] Terminated due to basis file overflow.
[6] Terminated due to branch and bound node table overflow (node table set to 200 problems).
[7] Input error due to file size (file has 193,481 lines).

Table 6. Comparisons of specialized algorithms with MPSX and OSL on an IBM 3081D (very large value of b in side constraint)

Problem Name	A100+1		A200+1		A300+1		A400+1	
Problem Characteristics								
Rows	201	201	401	401	601	601	801	801
Columns	4,000	4,000	16,000	16,000	36,000	36,000	64,000	64,000
Side Constraint Type	≤	=	≤	=	≤	=	≤	=
MPSX								
Nodes in BAB Tree	1	1,204	1	138	Too	Too	Too	Too
Iter	3,217	16,283	7,665	22,048	many	many	many	many
Time (Sec)	37	1,103	199	1,641	binary	binary	binary	binary
Obj. Value Best Inc.	3,852	7,030	3,691	NA	variables	variables	variables	variables
Status	Optimal	Feasible[1]	Optimal	Unknown[2]	Unknown[3]	Unknown[3]	Unknown[3]	Unknown[3]
OSL								
Nodes in BAB Tree	0	2,318	1	5	0	6	Error	Error
Iter	253	24,228	481	NA	794	NA	during	during
Time (Sec) ≥	16	3,530	100	NA	365	NA	input	input
Obj. Value Best Inc.	3,852	NA	3,691	NA	3,696	NA	NA	NA
Status	Optimal	Unknown[4]	Optimal	Unknown[5]	Optimal	Unknown[5]	Unknown[6]	Unknown[6]
ASSIGN+1								
Time (Sec)	0.11	1.26	0.52	5.20	1.17	13.69	3.16	26.82
Obj. Value	3,852	7,023	3,691	6,503	3,969	7,437	3,896	6,993
Dev. from Feasibility	0%	0.91%	0%	0.60%	0%	0.45%	0%	0.46%
Max. Dev. from Opt.	0%	3.27%	0%	0%	0%	1.68%	0%	1.79%

[1] Terminated due to node table overflow after obtaining the first incumbent but prior to obtainig an optimum.
[2] Manually stopped after 10 hours of wall clock time and 25 minutes of CPU prior to obtaining the first incumbent.
[3] MPSX/370 restricts the number of integer variables to maximum of 32,767.
[4] Terminated due to basis file overflow.
[5] Terminated due to OSL data overflow.
[6] Input error due to file size (file has 193,481 lines).

Table 7. Comparison of the integer and real versions of pure assignment codes. (The Weitek floating point accelerator was activated in all runs.)

Problem Size	400x400	800x800	1000x1000	1200x1200
SEMI (Secs.) (integer)	3.66	14.93	26.36	37.26
ASSIGN+1 (Secs.) (floating point)	4.00	16.38	28.37	40.71
Increase for floating point arithmetic	9.56%	9.71%	7.62%	9.25%

V. THE SINGLY CONSTRAINED UNBALANCED ASSIGNMENT PROBLEM

Navy personnel assignment problems are unbalanced in which the number of jobs n exceeds the number of men n. After dualizing the side constraint we obtain an unbalanced pure assignment problem whose dual is

$$\text{maximize} \quad \sum_i \lambda_i + \sum_j \pi_j \tag{10}$$

$$c_{ij} - \lambda_i - \pi_j \geq 0, \quad (i,j) \in E \tag{11}$$

$$\pi_j \leq 0, \qquad j = 1, \ldots, m. \tag{12}$$

The dual variable, π_j, is associated with job j and the dual variable, λ_i, is associated with the man i. The dual problem for the balanced assignment problem, (1)-(4) is (10) and (11).

SEMI, the FORTRAN code used to solve the pure assignment problems in the previous sections of this paper is the software implementation of a shortest augmenting path algorithm developed by Kennington and Wang [1990a]. The algorithm is a dual method and consists of four phases: column reduction, reduction transfer, row reduction augmentation, and shortest path augmentation. In each phase both dual feasibility, $c_{ij} - \lambda_i - \pi_j \geq 0$ for all $(i,j) \in E$ and complementary slackness $x_{ij}(c_{ij} - \lambda_i - \pi_j) = 0$ for all $(i,j) \in E$ are maintained and the procedure works toward obtaining primal feasibility, (2) and (3). Minor modifications to SEMI were incorporated so that $\pi_j \leq 0$ for all j was also maintained throughout the four phases. The modified code is called UNBAL_SEMI.

Table 8 presents our empirical results comparing SEMI and UNBAL_SEMI for the assignment problem and presents results for the singly constrained unbalanced assignment problem. Test runs were performed on an IBM 3081D and a Sequent Symmetry S81. Every entry in columns 2-6 of Table 8 is the average run time for three randomly generated problems except for the entries in the last row which are for a singly constrained unbalanced assignment problem provided by the Navy Personnel Research and Development Center in San Diego.

By adding dummy nodes and artificial arcs, one can always convert an unbalanced problem to a balanced one. As shown in Table 8, the specialized code for the unbalanced problem can run four times faster than the corresponding balanced code. For the 400x600 assignment problems, UNBAL_SEMI was four times faster than SEMI on problems in which 200 dummy men and 120,000 dummy arcs were appended. We also find that for this application, the IBM 3081D is approximately twice as fast as the Sequent Symmetry S81.

Table 8. CPU times (sec.) on an IBM 3081D and wall clock times (sec.) on a Sequent Symmetry S81 for balanced and unbalanced pure assignment problems and singly constrained unbalanced assignment problems.

Size	**Assignment**				**Assignment+1 Side Constraint**	
	SEMI[1]		UNBAL_SEMI		UNBAL_ASSIGN+1	
	IBM	Sequent	IBM	Sequent	IBM	Sequent
100x100	0.11	0.19	0.14	0.28	1.89	3.31
100x200	0.44	1.28	0.07	0.16	1.39	2.59
100x300	1.79	3.70	0.14	0.21	2.21	4.17
200x200	0.44	0.69	0.63	1.18	6.84	9.51
200x300	1.07	2.16	0.22	0.49	4.76	8.83
200x400	2.36	5.13	0.26	0.59	5.86	11.39
300x300	1.22	2.14	1.74	3.37	15.76	32.78
300x400	1.70	3.35	0.52	1.15	9.92	18.90
300x500	3.22	6.91	0.53	1.18	10.84	25.46
400x400	2.30	4.18	3.78	7.86	37.83	74.09
400x500	2.23	4.63	0.88	1.92	16.02	31.86
400x600	3.68	8.80	0.90	2.00	16.90	35.72
98x362[2]	NA	NA	NA	NA	0.28	0.59
Total	20.56	43.16	10.04	20.39	130.50	258.74

[1] dummy men nodes and artificial arcs are added to balance men and jobs.
[2] real problem provided by the Navy.

VI. SUMMARY AND CONCLUSIONS

We have presented a new algorithm, ASSIGN+1, for the singly constrained assignment problem. This algorithm is applicable for balanced and unbalanced assignment problems having either an inequality or an equality side constraint. The algorithm uses Lagrangean relaxation and solves a series of pure sparse assignment problems.

The empirical results of test runs of the FORTRAN implementation of the algorithm for balanced problems with inequality and equality side constraints and unbalanced problems with an inequality side constraint indicate that these three codes are very robust and need very few iterations to satisfy the optimality criteria. Remarkably, near optimal solutions to integer programs with over one-half million binary variables are obtained in less than fifteen minutes on a Sequent Symmetry S81 using a single processor. The results from test runs of ASSIGN+1, MPSX, and OSL proves the superiority of the new algorithms over state-of-the-art general purpose software.

REFERENCES

V. Aggarwal, "A Lagrangean-relaxation method for the constrained assignment problem," *Computers and Operations Research* vol. 12 pp. 97–106, 1985.

E. Allen, R. Helgason, J. Kennington, and B. Shetty, "A generalization of Polyak's convergence result for subgradient optimization," *Mathematical Programming* vol. 13 pp. 309–318, 1987.

M. Ball, U. Derigs, C. Hilbrand, and A. Metz, "Matching problems with generalized upper bound side constraints," *Networks* vol. 20 pp. 703–721, 1990.

R. Barr, K. Farhangian, and J. Kennington, "Networks with side constraints: an LU factorization update," *The Annals of the Society of Logistics Engineers* vol. 1 pp. 66–85, 1986.

J. Brans, M. Leclercq, and P. Hansen, "An algorithm for optimal reloading of pressurized water reactors," *Operational Research'72*, Editor M. Ross, North Holland Publishing Company: Amsterdam, 1973, pp. 417–428.

N. Bryson, "Parametric programming and Lagrangian relaxation: the case of the network problem with a single side-constraint," *Computers and Operations Research* vol. 18 pp. 129–140, 1991.

H. Everett, "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources," *Operations Research* vol. 11 pp. 399–417, 1963.

S. Gass and T. Saaty, "The computational algorithm for the parametric objective function," *Naval Research Logistics Quarterly* vol. 2 pp. 39–45, 1955.

A. Geoffrion, "An improved implicit enumeration approach for integer programming," *Operations Research* vol. 17 pp. 437–454, 1969.

A. Geoffrion, "Lagrangean relaxation for integer programming," *Mathematical Programming* vol. 2 pp. 82–114, 1974.

F. Glover, "A Multiphase-Dual Algorithm for the Zero-One Integer Programming," *Operations research* vol. 13 pp. 879–919, 1965.

F. Glover, D. Karney, D. Klingman, and R. Russell, "Solving Singly Constrained Transshipment Problems," *Transportation Science* vol. 12 pp. 277–297, 1978.

A. Gupta and J. Sharma, "Tree search method for optimal core management of pressurised water reactors," *Computers And Operations Research* vol. 8 pp. 263–269, 1981.

J. Kennington and Z. Wang, "An empirical analysis of the dense assignment problem", Technical Report 88-OR-16, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275, 1989.

J. Kennington and Z. Wang, "A shortest augmenting path algorithm for the semi-assignment problem," to appear in *Operations Research*, 1990a.

J. Kennington and Z. Wang, SEMI Users Guide, Technical Report 90-CSE-20, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275, 1990b.

J. Kennington and A. Whisman, "NETSIDE users guide," Technical Report 90-CSE-37, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275, 1990.

D. Klingman and R. Russell, "Solving constrained transportation problems," *Operations Research* vol. 23 pp. 91-106, 1975.

D. Klingman and R. Russell, "A stream lined approach to the singly constrained transportation problem," *Naval Research Logistics Quarterly* vol. 25 pp. 681-695, 1978.

Mathematical Programming System Extended, Mixed Integer Programming/370 Program Reference Manual, IBM SH19-1099-1, 1979.

J. Mazzola and A. Neebe, "Resource constrained assignment scheduling," *Operations Research* vol. 34 pp. 560-572, 1986.

G. Nemhauser and L. Wolesy, *Integer and Combinatorial Optimization*, John Wiley and Sons: New York, NY, 1988.

Optimization Subroutine Library: Guide and Reference, IBM, SC23-0519-1, 1990.

R. Parker and R. Rardin, *Discrete Optimization*, Academic Press Incorporated: New York, NY, 1988.

H. Salkin, *Integer Programming*, Addison-Wesley Publishing Company: Reading, Massachusetts, 1974.

J. Shapiro, "Generalized Lagrange multipliers in integer programming," *Operations Research* vol. 19 pp. 68-76, 1971.

J. Shapiro, "A survey of Lagrangean techniques for discrete optimization," *Annals of Discrete Mathematics* vol. 5 pp. 113-138, 1979.

Generalized Networks: Parallel Algorithms and an Empirical Analysis

R.H. Clark[1], J.L. Kennington[2], R.R. Meyer[1] and M. Ramamurti[2]

1 Center for Parallel Optimization
Computer Sciences Department
The University of Wisconsin-Madison
Madison, Wisconsin 53706 USA

2 Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275 USA

The objective of this research was to develop and empirically test new simplex-based parallel algorithms for the generalized network optimization problem. One of these algorithms is essentially a "data parallel" method in which each processor executes identical code on a portion of the data. (However, since the data sets are not necessarily disjoint, "locks" are used to ensure exclusive access.) A second algorithm exhibits "control parallelism", using different processors to simultaneously execute the different subtasks of the simplex method. "Locks" are not needed in this second approach, but, instead, at the beginning of each pivot, an "audit" of the proposed entering arc is performed in order to ensure correctness of the method. These parallel algorithms were implemented on the Sequent Symmetry multiprocessor, empirically tested on a variety of problems produced by two random problem generators. and compared with two leading state-of-the-art serial codes. Good speedups were obtained relative to the serial codes, and the relative performance of the two parallel methods was found to be dependent on connectedness properties of the optimal solutions of the test problems. The largest test problem, a generalized transportation problem having 30,000 nodes and 1.2 million arcs was optimized in approximately eleven minutes by our parallel code, displaying a speedup of 13 on 15 processors.

The *generalized network flow problem* (also called the *flow with gains model*) in its most general form is defined as follows:

$$\begin{aligned} \min_{x} \quad & cx \\ \text{s.t.} \quad & Gx = b \qquad\qquad \text{(GN)} \\ & 0 \le x \le u \end{aligned}$$

where G is an $m \times n$ matrix having at most two nonzero entries in each column, c is an n-vector of costs, b is an m-vector of right-hand-sides, and u is an n-vector of upper bounds. Associated with the matrix G is a generalized graph $[N, A]$, where N is a set of nodes and A is a set consisting of pairs of nodes (arcs) and (possibly) singletons (root arcs). The nodes correspond to the rows of G and the arcs and root arcs correspond to columns of G. (To simplify the discussion, we will use the term "arcs" to refer to both "ordinary" arcs (which connect two nodes) and root arcs, which are incident to only one node and correspond to columns of G with only one non-zero element.) As with the pure network flow problem, which we designate as PN, the simplex algorithm for GN can be executed on a graph. The graph of a basis for GN is a collection (forest) of one or more quasi-trees, where a quasi-tree is a tree with exactly one additional arc (making it either a rooted tree or a tree with exactly one cycle). This structural property plays an important role in one of the parallel algorithms described below. Figure 1 shows a forest of quasi-trees.

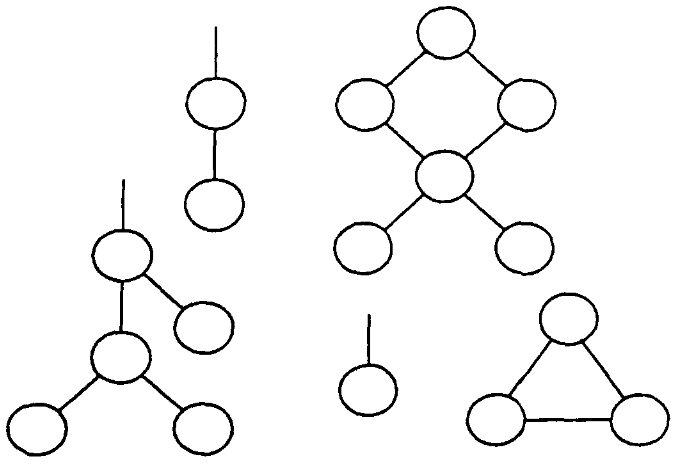

Figure 1 A Forest of Quasi-Trees

1. Survey and Overview

The generalized network model can be used to optimize network problems found in the areas of investment planning, job scheduling, pure network optimization and others. The applications are characterized by networks for which any arc may gain or lose flow at a linear rate assigned to that arc. Profit from interest or dividends can be modeled by a network with gains, and loss from evaporation or seepage can be modeled by a network with losses. A generalized network without gains or losses is a pure network. Further discussion of applications can be found in Glover et al. [15] and Mulvey and Zenios [23]. The graphical structure of a basis for G allows the use of labeling procedures for basis representation. Glover, Klingman, and Stutz [16] developed the first specialized primal simplex code (NETG) which exploited this graphical structure. Many theoretical and computational improvements have been made to this code over the last fifteen years (see Glover et al. [15]) and Elam et al. [13]). A similar implementation was also developed in Langley [22]. Adolphson and Heum [1] presented computational results with their generalized code which used an extension of the threaded index method of Glover et al. [17]. Brown and McBride presented the details of their generalized network code (GENNET) in [5]. Tomlin [26] developed the first assembly language code which is part of Ketron's MPS III system. Recently, other codes have been developed by Engquist and Chang [14]), Mulvey and Zenios [23]), and by Ali, Charnes, and Song [3]). The first parallel generalized code was developed by Chang, Enquist, Finkel, and Meyer [9]) for the Wisconsin CRYSTAL Multicomputer, and the second (see Clark and Meyer [11]) for the Sequent 21000, also at the University of Wisconsin. The first C language code is discussed in Nulty and Trick [24]. Another assembly language code is discussed in Chang, et al, [7]. The serial codes GENFLO, a modification of GENNET, and GRNET2 are discussed in Ramamurti [25] and Clark and Meyer [12], respectively. Computational results for these two codes will be given in Section 3. TPGRNET, a parallel code that assigns distinct tasks to different processes is discussed in Clark and Meyer [12] and additional results for this code are given in Section 3. A summary of this prior software may be found in Table I.

In Section 2, a brief discussion of some strategies for parallelizing the primal simplex method will be given, along with detailed discussions of two codes PGRNET and TPGRNET. PGRNET executes pivots and computes reduced costs in parallel. TPGRNET computes reduced costs in parallel and overlaps this with the serial execution of pivots.

Table I Survey of generalized network codes

Code	Language	Authors	Year
NETG	FORTRAN	Glover, F., Klingman, D. Stutz, J.	1973
	FORTRAN	Langley, W.	1973
	FORTRAN	Adolphson, D., Heum, L.	1981
GENNET	FORTRAN	Brown, G., McBride, R.	1984
GWHIZNET	ASSEMBLER	Tomlin, J.	1984
GRNET	FORTRAN	Engquist, M., Chang, M.	1985
LPNETG	FORTRAN	Mulvey, J., Zenios, S.	1985
	FORTRAN	Ali, I., Charnes, A. Song, T.	1986
GRNET-K (parallel)	FORTRAN	Chang, M., Engquist, M. Finkel, R., Meyer, R.	1987
PGRNET (parallel)	FORTRAN	Clark, R., Meyer, R. Chang, M.	1987
GNO/PC	C	Nulty, W., Trick, M.	1988
GRNET-A	ASSEMBLER	Chang, M., Cheng, M. Chen, C.	1988
GENFLO	FORTRAN	Ramamurti, M.	1989
GRNET2 (serial)	FORTRAN	Clark, R., Meyer, R. Chang, M.	1989
TPGRNET (parallel)	FORTRAN	Clark, R., Meyer, R.	1989

The test problems discussed in Section 3 are generated by 1) NETGEN [21], a pure network problem generator, 2) GNETGEN, a generalized network problem generator based on NETGEN, and 3) MAGEN [12], a generalized network problem generator based on GTGEN [8].

In Section 3, it is established that the two serial codes GRNET2 and GENFLO are competitive with GENNET, a state-of-the art generalized network code discussed in Brown and McBride [5]. This comparison is made by giving results for NETGEN and GNETGEN problems. Next, results are given for TPGRNET for a group of transshipment problems generated by GNETGEN with up to 6,000 nodes and up to 50,000 arcs. Results are then given for PGRNET and TPGRNET for a group of generalized transportation problems generated by MAGEN. All of these problems have 30,000 nodes and over 320,000 arcs. These problems are much more difficult than the GNETGEN problems in terms of both size and the pivots/nodes ratio. In addition, the variation in the granularity of the optimal solutions of these problems allows a good comparison to be made of the suitability of the two parallel approaches as a function of solution granularity.

2 SIMPLEX ALGORITHMS FOR GENERALIZED NETWORKS

2.1 Serial Primal Simplex Algorithms

In this section we briefly discuss the specialization of the primal simplex method for generalized networks. A more detailed discussion can be found in Kennington and Helgason [20] and Jensen and Barnes [19].

Input:

1. A generalized graph $[N, A]$.
2. A cost $c[a]$ and arc capacity $u[a]$ for each arc $a \in A$.
3. The constraint matrix G.
4. A right-hand-side value $b[n]$ for all $n \in N$.

Output:

1. The termination type indicator β and flow array $\bar{x}[a]$. ($\beta = 1$ implies that the problem is unbounded, $\beta = 2$ implies that the problem has no feasible solution, and $\beta = 3$ implies that the optimal solution is given in $\bar{x}[a]$.)

The primal simplex algorithm for generalized networks can be partitioned into three subroutines, PRICE, RATIO and UPDATE. These correspond to the computation of reduced costs, the ratio test, and the basis update in the simplex method for general LP's. Each of the subroutines makes use of the block diagonal (and nearly triangular) nature of the bases for (GN), as discussed in Adolphson [2] and Barr, Glover and Klingman [4]. The primal simplex algorithm can be summarized as follows:

```
Procedure SIMPLEX
begin
1.      $\beta \leftarrow 0$
2.      initialize duals ($\pi$)
3.      call module PRICE
4.      if $\beta \neq 0$, then terminate
5.      call module RATIO
6.      if $\beta \neq 0$ terminate
7.      call module UPDATE
8.      goto 3.
end
```

In module PRICE, reduced costs are computed for arcs (variables) by using the formula $r_{ij} = (c - \pi G)_{ij}$, where r_{ij} is the reduced cost for arc (i, j). The expression $(c - \pi G)_{ij}$ has at most three terms, since G has at most two non-zero entries in each column. The heuristics employed by GRNET2, PGRNET and TPGRNET to determine which arcs to price and to use as pivots will be discussed later, and involve maintaining candidate lists of pivot eligible arcs in very different ways. In fact, the parallel approaches, because of their asynchronous behavior, produce pivot sequences that would be difficult if not impossible to replicate in a serial algorithm. In module RATIO, the ratio test for a given pivot-eligible arc is performed by identifying the incident quasi-trees (i.e. the incident basis components) and following the path from each end of the arc to the (generalized) root of the corresponding quasi-tree(s). As this traversal is made, the flow on the basic arcs in the path is checked, and the arc with the "minimum ratio" is selected as the outgoing arc. In module UPDATE, flows as well as other tree data structures are updated.

GENFLO and GRNET2 are implementations of this algorithm, and they will be shown to be comparable in speed. However, the two codes differ in a few ways. GENFLO is a modification of GENNET, described in Brown and McBride [5] and it uses the GENNET pricing strategy. This strategy involves selecting a node and pricing out all of its incident arcs. GRNET2, on the other hand, scans its list of arcs linearly to locate pivot eligible arcs and doesn't attempt to focus on the arcs that are adjacent to some node. GRNET2 is specialized to solve problems for which at least one of the multipliers defined for each arc is equal to 1, while GENFLO allows each arc to have two arbitrary associated multipliers. The latter approach is desirable in integer programming applications since scaling variables to make one of the multipliers equal to 1 may destroy integrality. Also, reflecting an arc to convert a negative multiplier into a 1 requires that the arcs have upper bounds. Both GENNET and GRNET2 use the "little m" (or "gradual penalty") method [18] to find a feasible solution. Under this method, a moderate initial cost is given to the artificial arcs, and the resulting problem is approximately solved. Next, the cost on the artificial arcs is increased to create a new problem, and the optimal (or nearly optimal) basic feasible solution from the last problem is used as a warm start for the new one. This process of gradually increasing the cost on the artificial arcs and solving a sequence of "easy" problems can be shown empirically to yield a vast improvement over the "big M" method in terms of the total number of pivots required to solve a problem and in terms of the total CPU time. Some tests with GRNET2 have shown that the "little m" method is 29 times faster than the "big M" method for large problems having only one quasi tree in the optimal basis. GENFLO uses a simple closed-form expression to calculate the cost of the artificial arcs at each iteration. (The initial cost for the GENFLO artificial arcs is about 200. The cost on the artificial arcs is then roughly doubled at each step in the gradual penalty method.) The initial cost for the GRNET2 artificial arcs is 20 (assuming that the cost range for the regular arcs is 1-100). GRNET2 adds 5 to the cost on the artificial arcs after each step until the cost reaches 130, then for the next two steps the increment is by 10, and for the last two steps the increment is by 20. Finally, the cost is increased to "big M" and the problem is solved to optimality. An empirical comparison of these two codes with each other and with MPSX, the IBM general LP code, is given below.

2.2 Parallel Simplex Algorithms

In Clark and Meyer [11] an implementation of PGRNET is discussed. This code executes in parallel both pivots and pricing. Pivots are executed in parallel only if they involve updating separate quasi-trees (basis components). Even if the basis has only one component at optimality, this algorithm behaves quite efficiently during the beginning of the solution process, because the initial starting basis has as many components as nodes. PGRNET is used in Clark and Meyer [11] to solve problems generated randomly by GTGEN [8], and a code called MPGRNET is used to solve randomly generated multiperiod problems with a block diagonal structure generated by MPGEN [6]. MPGRNET reduces contention between processors by allocating specific quasi-trees to specific processors and allowing processors to execute pivots involving their quasi-trees (and only their quasi-trees) until they can find no "local" pivot eligible arcs. Optimality is then achieved by reverting to the PGRNET algorithm. Speedups for PGRNET in Clark and Meyer [11] range from 4.8 to 8.8 on 7 processors, and speedups for MPGRNET ranged from 8.8 to 36.9 on 12 processors. The superlinear speedup for some problems led the authors to develop a serial program that was much more efficient for the block diagonal, or "multi-period" problems. The resulting serial timing results yielded speedup results for MPGRNET that were slightly sublinear. An improved version of PGRNET is discussed in Clark and Meyer [12], and in Section 2.3 below.

A number of parallel algorithms for GN are discussed in Ramamurti [25]. The "Chaotic Column Partitioning Algorithm" (CCP) is similar to PGRNET in that it allows pivots to be executed in parallel, provided that the pivots involve updating separate quasi-trees. Another algorithm, known as the "Column Partitioning Algorithm" (CP) is similar to MPGRNET. The (CCP) algorithm, the (CP) algorithm and other algorithms are applied in Ramamurti [25] to problems generated randomly by GNETGEN, a modification of NETGEN [21]. Speedups for the (CCP) algorithm on 8 processors range from 1.27 to 1.73, and speedups for (CP) on 8 processors range from 1.33 to 2.48.

In Section 2.4 we discuss TPGRNET [12], an algorithm that devotes one processor to the task of executing pivots, and devotes all other processors to the task of computing reduced costs and managing candidate lists. (TPGRNET denotes "Task Parallel GRNET".) An algorithm (the Data Partitioning Algorithm) that is similar to TPGRNET in its partitioning of tasks is discussed in Ramamurti [25] and is applied to a set of generalized networks defined on grids. The advantage to having one processor do all pivoting is that there is no contention between processors for shared data structures, even when there is only one basis component in the optimal basis. This strategy yields an algorithm that is robust in the sense that it has a behavior that does not depend heavily on the nature of the optimal basis, and hence is appropriate for arbitrary generalized networks.

2.3 PGRNET (Parallel GRNET)

Figure 2 gives a flow chart for PGRNET. Parallel portions of the code are emphasized by parallel lines. "l.t." designates *list_threshold*, a candidate list parameter.

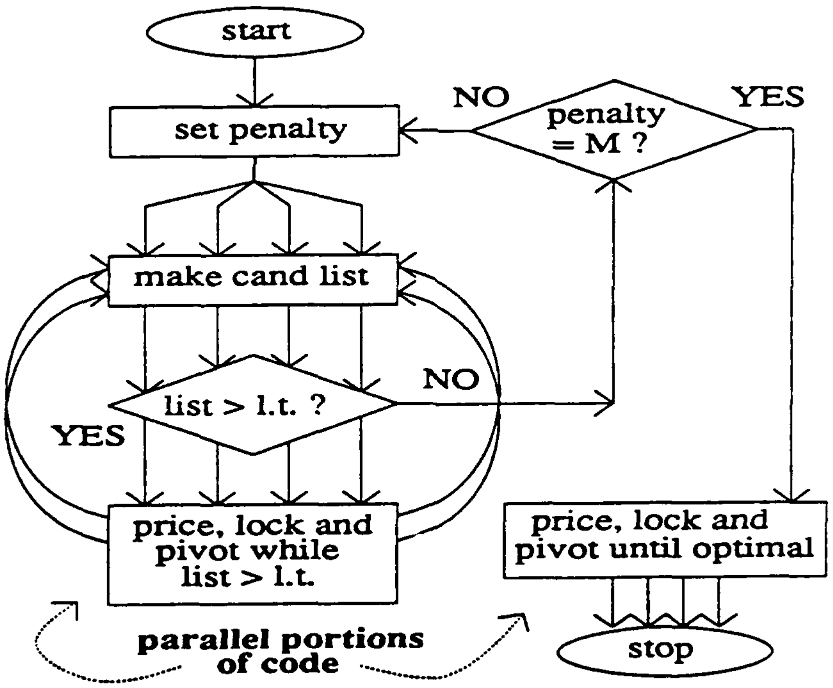
start
set penalty
NO
penalty = M ?
YES
make cand list
list > l.t. ?
NO
YES
price, lock and pivot while list > l.t.
price, lock and pivot until optimal
parallel portions of code
stop

Figure 2. Flow Chart For Parallel Algorithm PGRNET

The (parallel) PGRNET algorithm can be summarized as follows:

PGRNET

INITIALIZATION

In parallel, generate the initial flows on the artificial arcs. Divide the problem arcs into roughly equal-sized segments for pricing in the next stage.

STAGE 1 (*parallel pivoting with candidate lists*)

Asynchronously and in parallel scan the segments of the arc set to develop multiple candidate lists. Pivot arcs are selected from the candidate lists, and quasi-trees are "locked" before pivots are made. (In a shared memory environment, a locking operation by a processor on the portion of the shared data corresponding to the one or two selected quasi-trees serves to ensure exclusive access to those quasi-trees by the locking processor .) When, for a particular segment of the arc set, it is not possible to develop a candidate list with more than *list_threshold* entries, check the penalty on the artificial arcs. If this penalty has reached its maximum value go to STAGE 2. Otherwise, assign a new value to the penalty of the artificial arcs, update the duals in parallel and continue asynchronous pivoting.

STAGE 2 (*parallel verification of optimality*)

Scan the segments of the arc list in parallel to locate pivot-eligible arcs. If a pivot-eligible arc is found, lock the associated quasi-trees, and execute the pivot (if the quasi-trees were successfully locked). If an entire sweep through the segments can be made without finding any pivot-eligible arcs, optimality has been reached.

Arcs are partitioned roughly equally among segments. If there are n arcs and P segments, then processor 1 has arcs (1) through $\lceil n/P \rceil$, processor 2 gets arcs $\lceil n/P \rceil + 1$ through $\lceil 2n/P \rceil$ and so forth. (A more sophisticated allocation of the non-artificial arcs could be based on an approximation of the topology of the optimal solution. In the limiting case, if the optimal topology is known, lock contention could be reduced significantly by allocating to one processor all of the arcs between nodes of a given quasi-tree or group of quasi-trees. This approach could also be used to solve perturbed problems more efficiently by using the optimal topology of the base case.) The dual variables of all the nodes, the predecessor threads, the successor threads and all other tree functions required by the generalized network simplex method are stored in shared memory and are available to all processors. It is important to emphasize that only the acquisition of problem data (i.e., generating data or reading data) is done serially, and the solution process is entirely parallel. The number of partitions is equal to the number of processors, and all processors execute the same code, which is almost identical to that of GRNET2. The major differences in the parallel code are the quasi-tree locking, the "audit" of the reduced cost of the proposed entering arcs, and the distributed dual update. The asynchronicity of the quasi-tree locking and its effect on pivot selection make this parallel approach very difficult to emulate on a serial computer. We now present some additional detail on this parallel approach.

During a *parallel pivoting* stage, Stage 1, each processor makes its own candidate list of pivot- eligible arcs from the arcs assigned to it. These candidate lists are made in the same way that candidate lists are made in GRNET2. Each processor p chooses its next pivot arc from its candidate list by selecting the pivot-eligible arc with the greatest reduced cost in absolute value. If the quasi-trees at the ends of the arc have not been locked by another processor, p locks the quasi-trees to keep other processors from interfering with the tree update, checks the reduced cost, and, if the arc is still pivot eligible, performs the pivot and removes the arc from the candidate list. If the quasi-trees are already locked, or if the arc is no longer pivot eligible, processor p removes the arc from the candidate list and chooses another arc. The dual update part of the pivot operation has also been parallelized as follows: a processor traversing a quasi-tree to update duals puts roots of "large" subtrees that are encountered on a shared queue; other processors periodically check the queue, and when it is non-empty, take a subtree off the the queue and perform the dual update on the subtree. When the candidate list belonging to p has no more than *list_threshold* arcs, p develops a new candidate list. If the new candidate list also has no more than *list_threshold* entries, processor p sets a flag in shared memory to indicate that it is having difficulty finding pivot-eligible arcs. This flag is checked frequently by all processors, and when it is set, processor 1 checks to see if the penalty on the artificial arcs is *big* M. If the penalty is *big* M, all processors enter Stage 2. If the penalty is smaller, then the processors increment the penalty on the artificial arcs and cooperate in recomputing the dual variables. Then all processors develop new candidate lists.

Stage 2 of PGRNET corresponds to a *verification of optimality* stage. The verification of optimality is done in parallel, and all processors execute the same tasks. Optimality is achieved by performing any remaining pivots. Processors sweep through their segments looking for pivot-eligible arcs, but no candidate lists are developed. If processor p finds a pivot-eligible arc, it locks the quasi-trees at either end of the arc, executes the pivot, and indicates to the other processors that they must restart their sweep (by setting flags in a shared array). This restart mechanism is needed because a pivot executed by processor *p* might cause an arc owned by another processor to become pivot-eligible. If processor *p* finds that one of the trees at the ends of a pivot-eligible arc is locked, it sets the other processors restart flags and restarts its own sweep. Each processor checks its restart flag frequently during Stage 2, and when a processor finds that its flag has been set, it marks the arc in its segment that was last priced , and continues pricing. If the processor prices all of its arcs up to the marked arc without finding any to be pivot-eligible and without finding its restart flag to be set, that processor informs the others that none of its arcs are pivot-eligible. Optimality is reached when all processors make a sweep through their arcs without finding their restart flags set, and without finding any arcs that are pivot-eligible.

PGRNET is thus an example of data or uniform parallelism. The results in Section 3 show that uniform parallelism works well for generalized network flow problems in which the number of quasi-trees in the optimal solution is not too small.

2.4 TPGRNET (Task-Parallel GRNET)

This algorithm is divided into two stages, with 97% of all pivots executed in Stage 1. During Stage 1, different tasks are allocated to different processors (see Figure 3). One processor executes all pivots, one processor selects pivot arcs for the pivoting processor, and all other processors do pricing and place pivot eligible arcs into a shared candidate list to be scanned by the selecting processor. If a pivot requires updating a large quasi-tree, the pivoting processor can enlist the help of the pricing processors by putting the root nodes of subtrees on a queue. When these root nodes are detected by the pricing processors in the course of a periodic check, they take them off the queue and update the duals in the corresponding subtrees. Stage 2, in which only about 3% of the pivots are done, is analogous to that of the data-parallel PGRNET approach in that each processor scans a segment of the arc list belonging to that processor. As in PGRNET, when a processor finds a pivot eligible arc, it locks the quasi-trees at the ends of the arc, to temporarily exclude all other processors from modifying those quasi-trees, checks the reduced cost, and, if it is still pivot eligible, executes the pivot. We revert to the data parallel approach and develop no candidate lists in Stage 2 because very few pivot eligible arcs exist at this stage. More details of the algorithm are given below. Figure 3 illustrates the flow of information during Stage 1, and Figure 4 gives a flow chart for TPGRNET. In both figures, dotted arrows indicate the direction of the flow of information.

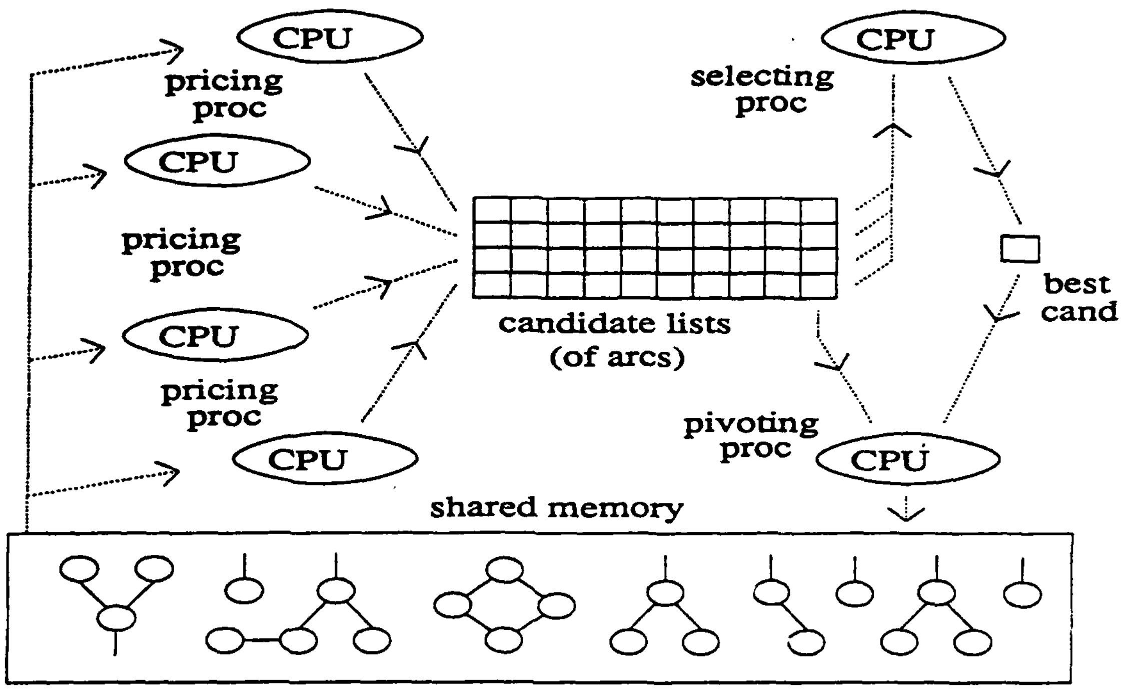

Figure 3 Flow of Information in TPGRNET, Stage 1

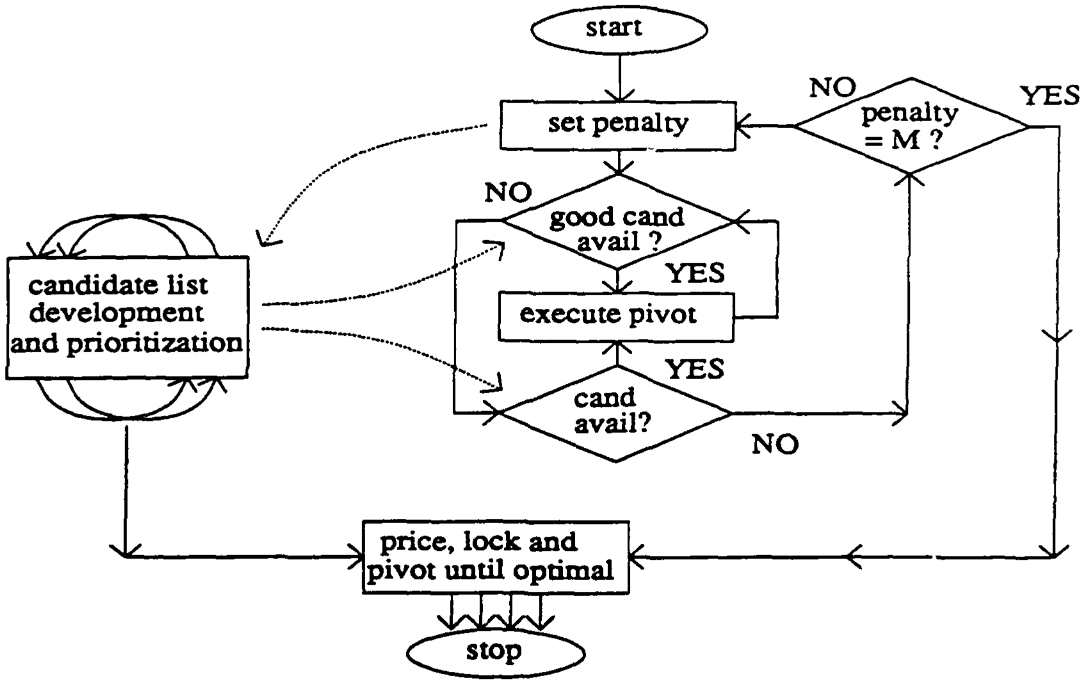

Figure 4 Flow Chart for Parallel Algorithm TPGRNET

The (parallel) TPGRNET algorithm is:

TPGRNET

INITIALIZATION

In parallel, generate the initial flows on the artificial arcs. Divide the problem arcs into roughly equal-sized segments for pricing during the next stage.

STAGE 1 (*parallel candidate list development overlapped with serial pivoting*)

A set of candidate lists is developed and prioritized in parallel. This process is continued during the pivot, which concurrently modifies some of the duals being used in candidate list development. When the pivot is completed, the next arc to enter the basis is selected by using the "best" arc from the candidate list (if this arc has a sufficiently good reduced cost) or a different arc (generally from the candidate lists–details are given below) if this is not possible. The latter case occurs very infrequently, and under conditions to be described below, may trigger an increase in the penalty cost or an exit to Stage 2.

STAGE 2 (*parallel pivoting and verification of optimality*)

Scan the segments of the arc list in parallel to locate pivot eligible arcs. If a pivot eligible arc is found, try to lock the associated quasi-trees, and, if the quasi-trees were successfully locked. execute the pivot. If an entire sweep through the segments can be made without finding any pivot eligible arcs, optimality has been reached.

We will now describe in detail the tasks performed by the individual processors during Stage 1 of TPGRNET. The pricing processors have the task of computing reduced costs and storing pivot eligible arcs in shared candidate lists of length 10. When processor p finds a pivot eligible arc, it recomputes the reduced cost of the first element in its candidate list to determine whether the new arc has a larger reduced cost in absolute value. If it does, the arc number gets stored in the first element of the array, and the previous entry is overwritten. Experience has shown that saving the previous entry yields no improvement in efficiency. If the new arc has a smaller reduced cost than the first arc in the candidate list, the new arc gets stored at a random location in the list. The pricing processors stay in a loop that includes three operations. First, there is the pricing operation. This uses most of the processor's CPU time. Second, there is a check to determine if the pivoting processor has put a subtree on the dual-update queue (because of space limits this is not shown in the figures). Third, there is a check to determine if Stage 1 of the algorithm has finished.

The pivot selecting processor has the task of scanning the candidate lists of the pricing processors to locate the pivot eligible arc with the largest reduced cost and storing that arc in a single shared variable called *best_cand*. This processor stays in a loop that has three operations. First, the processor checks whether *best_cand* is empty. If so, the processor looks in the first entry of each of the candidate lists to find an arc to put in *best_cand*. Second, the processor traverses the candidate lists to determine if there is a pivot eligible arc that has a reduced cost larger than the arc in *best_cand*. Third, there is a check to determine if Stage 1 of the algorithm is finished.

The pivoting processor stays in a loop in which it selects pivot arcs for itself (as described below), executes pivots, and directs the increases in the penalty on the artificial arcs. Whenever possible, the pivoting processor selects its pivot arc from *best_cand*, but before accepting an arc from *best_cand*, an "audit" is made to determine if the arc is still pivot eligible and if the reduced cost is sufficiently large in absolute value. (Note that the dual variables used by the pricing or selecting processor to price the arc may have been changed during the course of the pivot, so that an "audit" is needed to ensure that the proposed arc still provides a "good" pivot.) If the arc in *best_cand* has a small reduced cost, or if there is no arc in *best_cand*, the pivoting processor looks at the first entry of each of the candidate lists to find a pivot eligible arc. If a pivot eligible arc is found, the pivot is executed. If no pivot eligible arc is found, then either the penalty on the artificial arcs is increased, or Stage 2 is begun (if the penalty has reached *big M* and cannot be increased). The pivoting processor also has the task of directing the parallel update of dual variables during the execution of pivots and after the penalty on the artificial variables has been increased. During both of these operations, the pivoting processor can put the root nodes of subtrees onto the dual update queue, and the pricing processors will then assume the tasks of updating the duals on those subtrees.

3 COMPUTATIONAL EXPERIENCE

3.1 Results for pure network problems

Pure networks are a special case of generalized networks (all multipliers have a magnitude of 1). In order to compare the efficiency of general versus specific codes, we consider the relative performance of a general simplex code (MPSX), two generalized network codes, and a pure network code on a class of pure network test problems. Table II gives results for a collection of problems generated by NETGEN [21]. The problem numbers have the prefix "N", to indicate that they were generated by NETGEN, and a numerical suffix that indicates the standard NETGEN problem number [21]. All times are in seconds, and all runs were made on an IBM 3081-D24. Although the number of pivots executed by MPSX (the IBM proprietary mathematical programming system) is roughly equal to the number of pivots executed by NETFLO [20], NETFLO is roughly 68 times faster than MPSX due to the fact that it is designed to solve pure network problems, and it utilizes the tree structure of bases and uses integer arithmetic. GENNET uses an improved pricing strategy that reduces the total number of pivots by a factor of three, compared to MPSX. Overall, GENNET timings are about 54 times faster than MPSX. The timings and the number of pivots for GENFLO are similar to those of GENNET. GENFLO solves these problems with fewer pivots than GENNET, but CPU times are about 25% slower, possibly due to the fact that GENFLO allows for two arbitrary multipliers. *These results clearly justify the utility of specialized generalized network software.*

Table II Serial results for NETGEN problems (IBM 3081-D24)

	Size		MPSX		GENFLO		GENNET		NETFLO	
Problem	nodes	arcs	pivots	secs.	pivots	secs.	pivots	secs.	pivots	secs.
N15	400	4,500	2,818	30.60	1,288	1.41	1,307	1.19	2,073	0.47
N18	400	1,306	2,077	12.00	593	0.49	578	0.39	1,079	0.24
N19	400	2,443	4,229	29.40	688	0.71	765	0.53	1,305	0.23
N22	400	1,416	3,052	18.00	613	0.52	504	0.33	1,284	0.29
N23	400	2,836	7,073	57.60	492	0.47	604	0.45	1,156	0.22
N26	400	1,382	4,286	24.60	511	0.42	500	0.27	917	0.14
N27	400	2,676	11,829	95.40	628	0.55	826	0.46	1,730	0.28
N28	1,000	2,900	3,313	38.40	1,487	1.39	1,732	1.24	3,524	0.93
N29	1,000	3,400	3,744	43.80	1,889	1.59	1,996	1.18	4,570	1.12
N30	1,000	4,400	4,954	60.00	1,947	1.87	1,969	1.31	4,346	1.04
N31	1,000	4,800	6,232	81.00	2,171	2.13	2,347	1.47	4,798	1.13
N33	1,500	4,385	5,836	103.20	2,645	2.83	2,521	2.01	6,113	2.16
N34	1,500	5,107	6,503	110.40	2,498	2.50	2,943	2.10	7,640	2.37
N35	1,500	5,730	7,026	115.80	3,017	3.35	3,310	2.82	7,384	2.30
Total			72,972	820.20	20,467	20.23	21,902	15.75	47,919	12.92

3.2 Results for GNETGEN generalized network problems

NETGEN has been modified by D. Klingman to generate generalized network flow problems. The modified generator is called GNETGEN. Table III gives most of the GNETGEN input data for the small test problems G1 through G7. The prefix "G" for these problems indicates that they were generated by GNETGEN. The numerical suffix corresponds to the problem numbers in Table 2.2 in Ramamurti [25]. (The random seed for all of these problems is 13502460.)

Table III Input data for small GNETGEN problems

Problem	G1	G2	G3	G4	G5	G6	G7
Nodes	200	200	200	300	400	400	1,000
Arcs	1,500	4,000	6,000	4,000	5,000	7,000	6,000
Sources	100	5	15	135	20	30	20
Sinks	100	195	50	165	100	50	100
Cost Range	1-100	1-100	1-100	1-100	1-100	1-100	1-100
Gain Range	.5-1.5	.5-1.5	.25-.95	.5-1.5	.3-1.7	.5-1.5	.4-1.4
Supply	100k	100k	100k	100k	100k	100k	200k
% Capacitated	0	100	100	0	0	100	100
Bound Range	—	1-2k	1-2k	—	—	1-2k	4-6k

Table IV gives results for MPSX, GENNET and GENFLO for problems G1 through G7. For these problems, GENNET is about 12 times faster than MPSX and GENFLO about 11 times faster. GENFLO solves these problems with about 40% fewer pivots than MPSX. *Note that relaxing the assumption that one of the multipliers is unity results in an increase in computing time of only about 10%.*

Table IV Serial results for small GNETGEN problems (IBM 3081-D24)

	Size		MPSX		GENFLO		GENNET	
Prob.	nodes	arcs	pivots	secs.	pivots	secs.	pivots	secs.
G1	200	1,500	1,151	7.80	533	0.95	590	0.62
G2	200	4,000	550	3.00	358	0.23	443	0.22
G3	200	6,000	2,058	18.60	954	1.53	1,448	2.07
G4	300	4,000	4,112	47.40	2,106	4.23	2,703	3.50
G5	400	5,000	1,870	26.20	897	2.23	1,229	2.06
G6	400	7,000	1,408	16.80	1,171	1.68	1,591	1.59
G7	1,000	6,000	2,811	40.20	2,352	3.60	3,160	3.30
Total			13,960	160.00	8,371	14.45	11,164	13.36

Tables V through VII give the input data for the larger GNETGEN problems G8 through G22. These problems are generated with the same input data as problems 1 through 15 in Table 4.1a and 4.1b in Ramamurti [25]. The problems are grouped according to *rectangularity ratios* (arcs/nodes).

Table V GNETGEN problems G8-G13

	Problems					
Characteristics	**G8**	**G9**	**G10**	**G11**	**G12**	**G13**
Nodes	2,000	2,000	2,000	4,000	4,000	4,000
Arcs	13,000	13,000	13,000	26,000	26,000	26,000
Sources	150	150	150	150	150	150
Sinks	600	600	600	600	600	600
% Capacitated	100	50	0	100	50	0
Cost Range	1-100	1-100	1-100	1-100	1-100	1-100
Bound Range	1k-2k	1k-2k	—	1k-2k	1k-2k	—
Mult. Range	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5

Table VI GNETGEN problems G14-G16

	Problems		
Characteristics	**G14**	**G15**	**G16**
Nodes	6,000	6,000	6,000
Arcs	39,000	39,000	39,000
Sources	150	150	150
Sinks	600	600	600
% Capacitated	100	50	0
Cost Range	1-100	1-100	1-100
Bound Range	1k-2k	1k-2k	—
Bound Range	0.5-1.5	0.5-1.5	0.5-1.5

Table VII GNETGEN problems G17-G22

	Problems					
Characteristics	**G17**	**G18**	**G19**	**G20**	**G21**	**G22**
Nodes	2,000	2,000	2,000	2,000	2,000	2,000
Arcs	25,000	25,000	25,000	50,000	50,000	50,000
Sources	150	150	150	150	150	150
Sinks	600	600	600	600	600	600
% Capacitated	100	50	0	100	50	0
Cost Range	1-100	1-100	1-100	1-100	1-100	1-100
Bound Range	1k-2k	1k-2k	—	1k-2k	1k-2k	—
Mult. Range	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5

Table VIII gives a comparison of GENFLO and GRNET2 for problems G8 through G22. The two programs give ***nearly the same performance for these test problems***, despite the fact that the two codes have very different pricing strategies. The number of pivots for GRNET2 is about 15% less than for GENFLO, and the total time for GRNET2 on the test problem set is about 4% less.

Table VIII Serial results for large GNETGEN problems (Sequent S81)

	Size		GENFLO		GRNET2	
Problem	nodes	arcs	pivots	time	pivots	time
G8	2,000	13,000	5,108	60.5	3,892	51.9
G9	2,000	13,000	4,454	44.2	3,998	46.8
G10	2,000	13,000	4,634	50.2	3,808	48.5
G11	4,000	26,000	9,897	125.0	7,690	121.9
G12	4,000	26,000	9,815	123.6	7,460	115.1
G13	4,000	26,000	9,145	99.3	7,375	105.2
G14	6,000	39,000	13,262	145.4	10,245	142.2
G15	6,000	39,000	12,900	126.3	10,059	158.9
G16	6,000	39,000	13,653	141.0	10,456	152.0
G17	2,000	25,000	6,186	89.2	6,369	98.0
G18	2,000	25,000	6,596	93.4	5,260	78.4
G19	2,000	25,000	6,629	119.2	5,440	81.3
G20	2,000	50,000	8,601	198.4	9,608	194.8
G21	2,000	50,000	9,208	204.5	10,343	194.9
G22	2,000	50,000	8,500	174.2	7,913	133.8
Total			128,588	1,794.4	109,916	1,723.7

Table IX gives CPU times for TPGRNET (run on various numbers of processors) for problems G8 through G22. TPGRNET is faster than the parallel versions of GENFLO for these test problems, and it is more robust in the sense that CPU times usually decrease monotonically as the number of processors is increased. Hence, we report here only the performance results for TPGRNET. The column heading "cap" denotes the percentage of capacitated arcs, and the column heading "qtree" denotes the number of quasi-trees in the optimal solution. The serial times reported in the table are taken from GRNET2 if they have the "T" prefix, and they are taken from GENFLO if they have the "O" prefix. The serial time given is always taken from the faster of the two codes. The time totals from the bottom of Table IX are graphed in Figure 5.

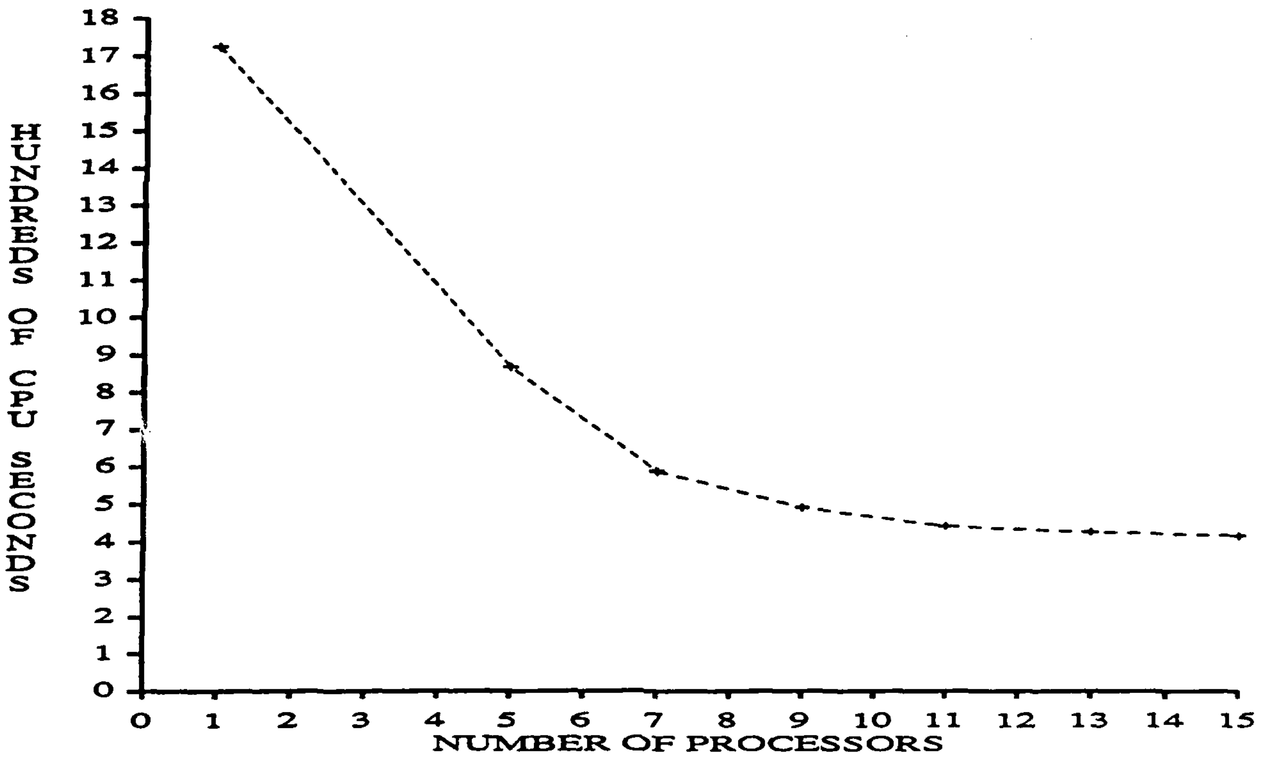

Figure 5 Composite results for TPGRNET

Table IX TPGRNET timings for problems G8 through G22

					Number		of	Processors		(Sequent)	
Prob	nds	arcs	cap	qtree	1	5	7	9	11	13	15
G8	2k	13k	100	4	T51.9	29.3	19.8	16.2	16.3	16.9	15.8
G9	2k	13k	50	1	O44.2	24.6	16.7	14.0	12.6	13.1	13.8
G10	2k	13k	0	2	T48.5	25.2	17.5	14.2	12.9	13.2	13.3
G11	4k	26k	100	3	T121.9	58.3	43.8	34.3	35.6	33.8	31.8
G12	4k	26k	50	1	T115.1	61.2	41.1	36.4	33.5	31.6	31.4
G13	4k	26k	0	3	O99.3	56.8	38.7	34.0	29.0	29.9	29.6
G14	6k	39k	100	5	T142.2	87.5	58.4	50.3	44.3	45.1	43.8
G15	6k	39k	50	7	O126.3	87.0	65.5	51.6	48.1	46.5	47.0
G16	6k	39k	0	2	O141.0	89.5	61.3	51.3	45.8	40.8	40.8
G17	2k	25k	100	9	O89.2	39.0	25.2	20.2	19.2	18.1	16.8
G18	2k	25k	50	3	T78.4	41.3	29.8	22.5	22.5	21.2	19.0
G19	2k	25k	0	3	T81.3	41.8	25.7	21.9	18.4	18.1	18.2
G20	2k	50k	100	2	T194.8	77.5	49.3	44.3	35.0	34.8	33.0
G21	2k	50k	50	3	T194.9	83.4	50.8	42.2	36.5	31.7	33.5
G22	2k	50k	0	1	T133.8	64.4	44.5	37.9	32.1	32.0	27.9
Totals					1723.5	866.8	588.1	491.3	441.6	426.8	415.7

Table X gives speedup results for problems G8 through G22, and the results are graphed for problems G14, G15, G16 and problems G20 ,G21, and G22. Although the percentage of capacitated arcs has little effect on speedup for this problem class, the rectangularity ratio is important in terms of efficiency. Problems G8-16 have the *smallest ratio of arcs to nodes*, and TPGRNET yields the smallest speedup for these problems. On the other hand, TPGRNET has an average speedup of 5.4 on 15 processors for problems G20 , G21 and G22, which have the *largest* arcs/nodes ratio. *Since TPGRNET achieves most of its parallelism from pricing arcs in parallel, the dependence of the efficiency of TPGRNET on the arcs/nodes ratio is understandable.*

Table X TPGRNET speedups for problems G8 through G22

					Number	of	Processors	(Sequent)			
Prob	nds	arcs	cap	qtree	1	5	7	9	11	13	15
G8	2k	13k	100	4	1.0	1.7	2.6	3.2	3.1	3.0	3.2
G9	2k	13k	50	1	1.0	1.7	2.6	3.1	3.5	3.3	3.2
G10	2k	13k	0	2	1.0	1.9	2.7	3.4	3.7	3.6	3.6
G11	4k	26k	100	3	1.0	2.0	2.7	3.5	3.4	3.6	3.8
G12	4k	26k	50	1	1.0	1.8	2.8	3.1	3.4	3.6	3.6
G13	4k	26k	0	3	1.0	1.7	2.5	2.9	3.4	3.3	3.3
G14	6k	39k	100	5	1.0	1.6	2.4	2.8	3.2	3.1	3.2
G15	6k	39k	50	7	1.0	1.4	1.9	2.4	2.6	2.7	2.6
G16	6k	39k	0	2	1.0	1.5	2.3	2.7	3.0	3.4	3.4
G17	2k	25k	100	9	1.0	2.2	3.5	4.4	4.6	4.9	5.3
G18	2k	25k	50	3	1.0	1.8	2.6	3.4	3.4	3.6	4.1
G19	2k	25k	0	3	1.0	1.9	3.1	3.7	4.4	4.4	4.4
G20	2k	50k	100	2	1.0	2.5	3.9	4.3	5.5	5.5	5.9
G21	2k	50k	50	3	1.0	2.3	3.8	4.6	5.3	6.1	5.8
G22	2k	50k	0	1	1.0	2.0	3.0	3.5	4.1	4.1	4.7
Averages					1.0	1.9	2.9	3.5	3.8	4.0	4.1

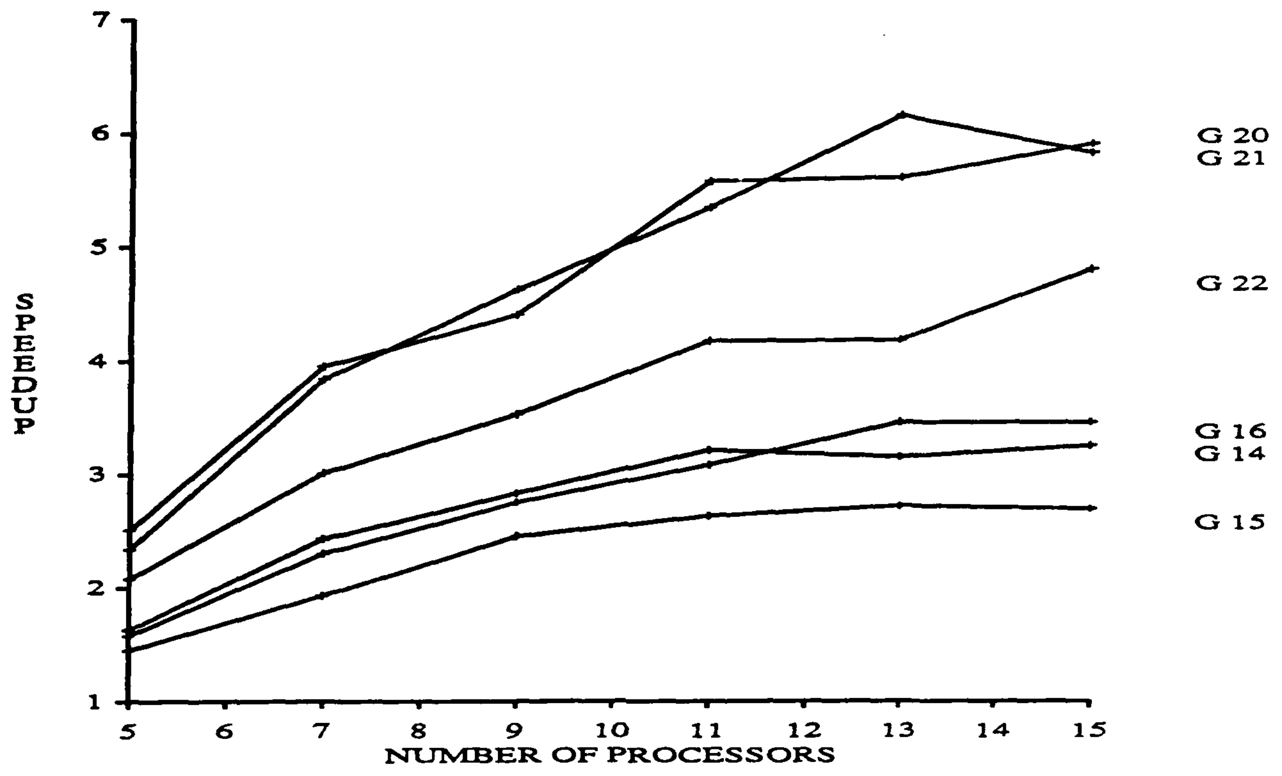

Figure 6 Speedups for TPGRNET

3.3 Results for MAGEN Problems

Table XI gives results for a group of large problems generated by MAGEN, the generator used in Clark and Meyer [12]. This is a modification of GTGEN, a generator described in Chang and Engquist [8]. All problems have 30,000 nodes and more than 300,000 arcs. The precise MAGEN input data for these problems, as well as optimal objective function values are given in Clark [10]. MAGEN generates random bipartite generalized network problems , *but allows the user to specify, roughly, the granularity of the generated problem,* (i.e., the user may adjust the number of quasi-trees in the optimal basis). Since problems 4.00 through 4.50 have very different granularities, the effect of granularity on the efficiency of TPGRNET and PGRNET can be studied by solving these problems. Although MAGEN was motivated by the need to compare parallel methods whose performance was dependent on test problem structue, it should also be noted that even from a serial computing viewpoint, granularity has a significant impact on solution time. For example, GRNET2 solves 4.50 sixteen times faster than 4.00, even though it does only 30% fewer pivots. This means that the pivots in 4.50 are relatively fast, due to the fact that quasi-trees are smaller on average than those of problem 4.0. PGRNET yields an impressive speedup of 11.1 over GRNET2 for 4.50, because the quasi-trees are numerous in the optimal basis (and in the intermediate bases). The serial version of GENFLO outperforms GRNET2 on problem 4.50 in terms of CPU time, but *GRNET2 significantly outperforms GENFLO for the more difficult problems in terms of both CPU time and the number of pivots.* Looking at problem 4.00, one sees that the fastest serial algorithm is GRNET2, and the fastest parallel algorithm is TPGRNET. The shared candidate list and parallel pricing strategy of TPGRNET makes TPGRNET outperform PGRNET in terms of the *number of pivots* for all problems, but, except for problem 4.00, this cannot compensate for the parallel pivoting of PGRNET. *In all of the other problems, PGRNET outperforms TPGRNET in terms of CPU time because PGRNET executes pivots in parallel.*

Table XI Results for PGRNET, TPGRNET, GENFLO, and GRNET2

Problem #	**4.00**	**4.01**	**4.03**	**4.05**	**4.10**	**4.50**
# qtrees at optimality	1	139	459	776	1,490	7,376
# nodes	30,000	30,000	30,000	30,000	30,000	30,000
# arcs	322,289	322,428	322,748	323,065	323,779	329,665
# pivots GENFLO	***	411,720	337,907	313,982	289,937	244,635
# pivots GRNET2	328,711	347,420	320,937	308,284	288,856	228,819
# pivs 19 procs PGRNET	365,633	383,483	345,325	326,323	300,496	231,507
# pivs 19 procs TPGRNET	265,774	305,979	288,279	272,322	263,411	221,544
CPU secs. GENFLO	***	36,523	10,524	5,121	2,436	1,038
CPU secs. GRNET2	22,434	9,679	4,525	3,096	2,340	1,388
CPU: 19 procs PGRNET	5,829	1,527	589	364	223	124
CPU: 19 procs TPGRNET	3,390	2,571	1,177	721	470	211
Speedup PGRNET	3.8	6.3	7.6	5.2	10.4	11.1
Speedup TPGRNET	6.6	3.7	3.8	4.2	4.9	6.5

*** Did not finish after 14 hours.

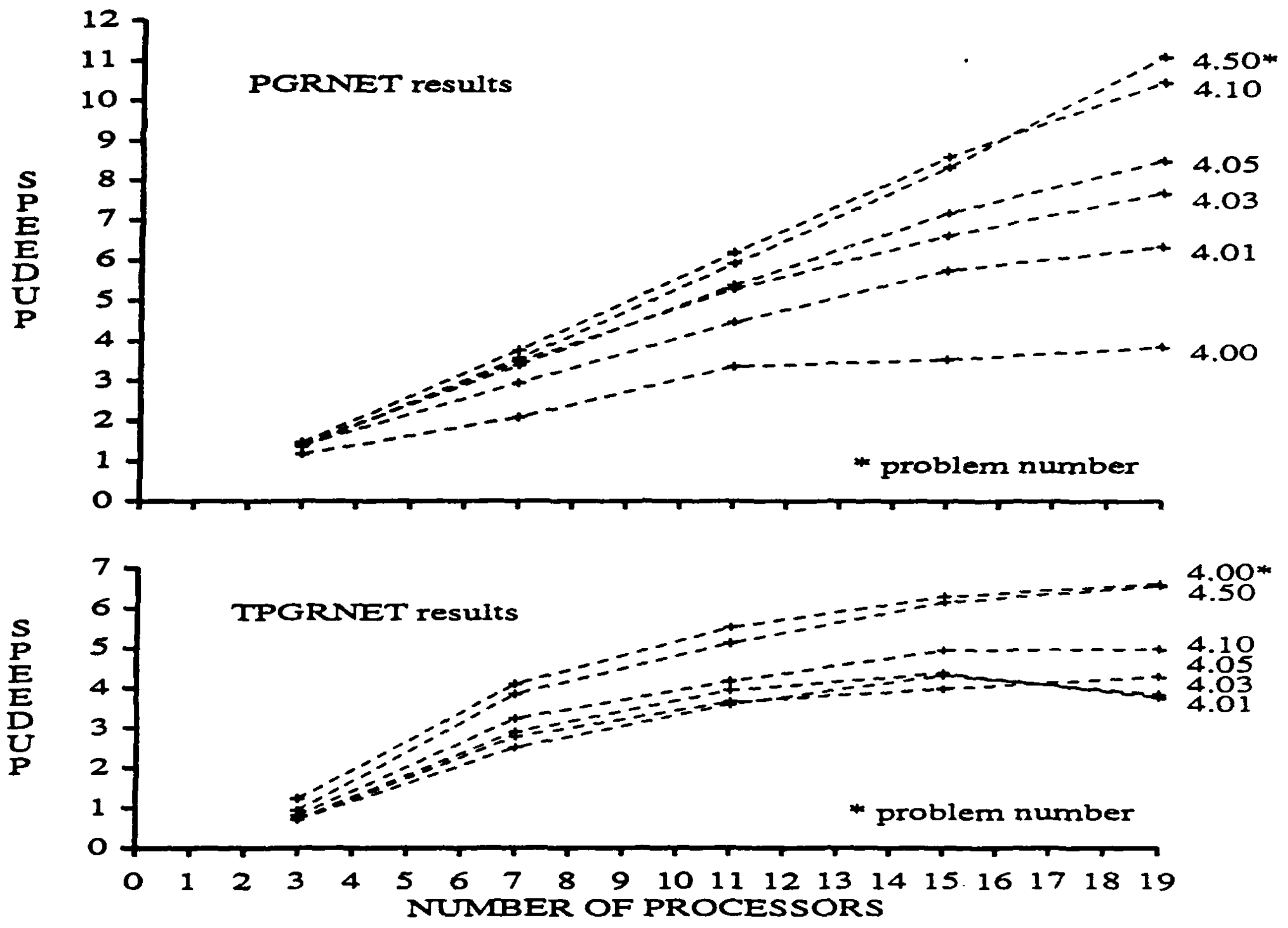

Figure 7 Speedups for PGRNET and TPGRNET for MAGEN problems

Figure 8 and Tables XII and XIII show results for two problems with more than a million variables. The problem reported in Table XII has tighter capacity constraints than does the Table XIII problem. *Both of these problems are small grained, so they can be solved quite efficiently by PGRNET.* The best speedup was achieved on the tightly constrained problem for which a speedup of 13 was achieved using 15 processors. Note that the tightly constrained problem was considerably more difficult to solve with the serial version of the code, so that there was more potential for improvement with a parallel approach.

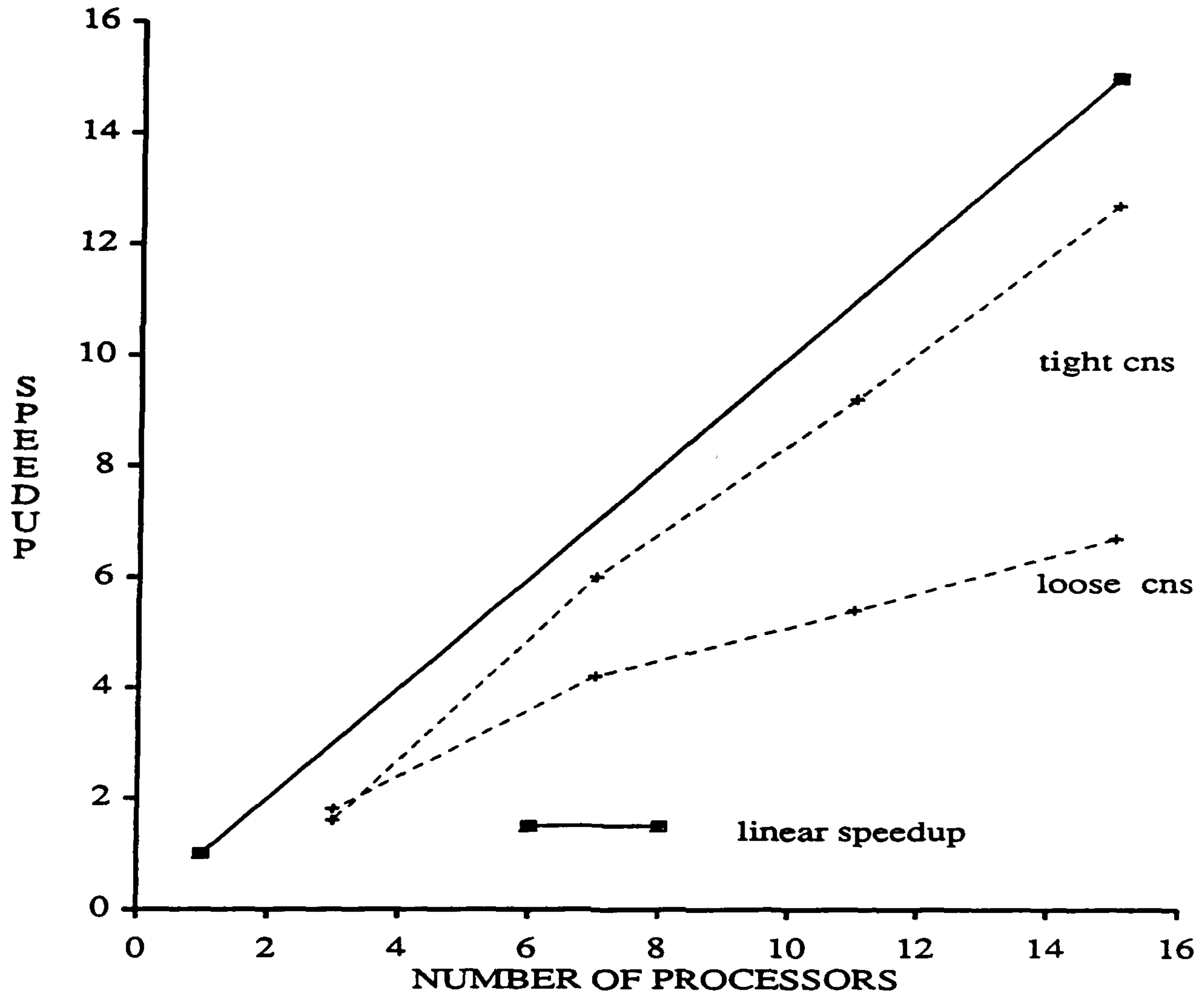

Figure 8 Speedups for PGRNET (# arcs > 1,000,000)

Table XII PGRNET results for tightly constrained problem

program	# arcs	# nodes	# qtrees	time	pivots	maj page swap
serial	1,267,185	30,000	14,859	8,415	706,776	914,478
15 procs	1,267,185	30,000	14,859	660	715,168	101,866

Table XIII PGRNET results for loosely constrained problem

program	# arcs	# nodes	# qtrees	time	pivots	maj page swap
serial	1,267,185	30,000	14,859	3,305	184,379	363,755
15 procs	1,267,185	30,000	14,859	490	186,969	45,672

4. SUMMARY AND CONCLUSIONS

The availability of powerful parallel computers has generated widespread interest in the development of new optimization algorithms for such machines. The parallel algorithms and software reported in this investigation demonstrate the effectiveness of these advanced computers for the optimization of generalized networks. Although these methods were developed and tested for a particular multiprocessor system with shared memory (Sequent Symmetry S81), they can be used with any shared memory parallel processing system.

In our empirical study we found that our serial network software is *at least forty times faster than MPSX for pure network problems and is at least an order of magnitude faster than MPSX for generalized networks.* We also demonstrated that relaxing the restriction that at least one of the multipliers associated with an arc be +1 results in an additional computational expense of only ten percent.

We believe that the best current serial software for these problems is GENNET [5] and GRNET2 [12] and we began our study of parallel algorithms with these codes. GENFLO (a two-multiplier version of GENNET) and GRNET2 provided the best single processor times for the empirical analysis presented in this study. For a comparison with parallel codes, both codes were run and the smaller serial time was used in speedup calculations for the parallel codes. In order to assess the effectiveness of our parallel approaches, we considered generalized networks with a large range of connectedness in optimal solutions, since this structure was a key factor in parallel efficiency. *The data-parallel PGRNET method proved to be more efficient on problems with many quasi-trees in the optimal solution, while the control-parallel TPGRNET algorithm was more effective on problems with relatively few quasi-trees.* The best speedup was achieved on a tightly constrained problem having 30,000 nodes and over 1.2 million arcs. For this problem PGRNET achieved a speedup of thirteen on fifteen processors. Clearly, the lower speedups reported here for problems with small numbers of quasi-trees in the optimal solutions indicate that both parallel approaches encountered difficulties in such cases in terms of scale-up to larger numbers of processors, in the sense that little improvement is demonstrated as the number of processors increases beyond 10. One extension that we are currently investigating to more effectively utilize additional processors in a control-parallel approach is to employ them as "ratio processors" that perform ratio tests (or approximations to ratio tests) on candidate arcs generated by the pivoting processors. Particularly in the case of difficult problems in which the number of pivots is large relative to the number of nodes and arcs, this approach holds the promise of increasing speedups by significantly reducing the total number of pivots. Another future area of research that we will be pursuing is the investigation of the extension of the control-parallel approach to general linear programs. Clearly, the allocation of tasks to processors for the general simplex method can be done in the same manner as TPGRNET. Relevant issues include the effect of increased pricing and pivoting times for general linear programs, alternatives to full ratio tests in the case that ratio processors are utilized, and the effect of increased data dependency of the pricing and pivoting tasks.

ACKNOWLEDGMENTS

This research was supported in part by NSF grants CCR-8709952 and CCR-8907671, the Air Force Office of Scientific Research under grants AFOSR-87-0199 and 89-0410 , the Department of Defense under contract number MDA 903-86-C0182, and the Office of Naval Research under contract number N00014-87-K-0223.

References

1. D. Adolphson and L. Heum, 1981. *Computational experiments on a threaded index generalized network code*, **ORSA/TIMS Joint National Meeting** Houston, Texas.

2. D. Adolphson, 1982. *Design of primal simplex generalized network codes using a preorder thread index*, Working Paper, School of Management, Brigham Young University, Provo, Utah.

3. I. Ali, A. Charnes and T. Song, 1986. *Design and implementation of data structures for generalized networks*, **Journal of Information and Optimization Sciences** 7 81-104.

4. R. Barr, F. Glover and D. Klingman, 1979. *Enhancements of spanning tree labeling procedures for network optimization*, **INFOR 17**, 16-34.

5. G. Brown and R. McBride, 1984. *Solving generalized networks*, **Management Science 30**, 1497-1523.

6. M. Chang, 1986. *A parallel primal simplex variant for generalized networks*, Ph.D. thesis, University of Texas at Austin.

7. M. Chang, M. Cheng and C. Chen, 1988. *Implementation of new labeling procedures for generalized networks*, **Technical Report**, Department of CS/OR, North Dakota State University, Fargo, North Dakota.

8. M. Chang and M. Engquist, 1986. *A parallel algorithm for generalized networks*, **Annals of Operations Research 14**, 125-145.

9. M. Chang, M. Engquist, R. Finkel and R. Meyer, 1988. *On the number of quasi-trees in an optimal generalized network basis*, **COAL Newsletter 14**, 5-9.

10. R. Clark, 1989. *The efficient parallel solution of generalized network flow problems*, Ph.D. thesis, University of Wisconsin-Madison.

11. R. Clark and R. Meyer, 1987. *Multiprocessor algorithms for generalized network flows*, **Technical Report #739**, Department of Computer Sciences, University of Wisconsin-Madison.

12. R. Clark and R. Meyer, 1990 *Parallel arc-allocation algorithms for optimizing generalized networks*, **Annals of Operations Research 22**, 129-160.

13. J. Elam, F. Glover and D. Klingman, 1979. *A strongly convergent primal simplex algorithm for generalized networks*, **Mathematics of Operations Research 4**, 39-59.

14. M. Engquist and M. Chang, 1985. *New labeling procedures for the basis graph in generalized networks*, **Operations Research Letters 4**, 151-155.

15. F. Glover, J. Hultz, D. Klingman and J. Stutz, 1978. *Generalized networks: A fundamental computer based planning tool*, **Management Science 24**, 1209-1220.

16. P. Glover, D. Klingman and J. Stutz, 1973. *Extension of the augmented predecessor index method to generalized network problems*, **Transportation Science 7**, 377-384.

17. F. Glover, D. Klingman and J. Stutz, 1974. *The augmented threaded index method for network optimization*, **INFOR 12**, 293-298.

18. M. Grigoriadis, 1984. *An efficient implementation of the network simplex method.* **Mathematical Programming Study 26**, 83-111.

19. P. Jensen and J. Barnes, 1980. *Network Flow Programming*, John Wiley and Sons, New York.

20. J. Kennington and R. Helgason, 1980. *Algorithms for Network Flow Programming*, John Wiley and Sons, New York.

21. D. Klingman, A. Napier and J. Stutz, 1974. *NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow problems*, **Management Science 20**, 814-821.

22. W. Langley, 1973. *Continuous and integer generalized flow problems*, Ph.D. thesis, Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

23. J. Mulvey and S. Zenios, 1985. *Solving large scale generalized networks,* **Journal of Information and Optimization Sciences 6**, 95-112.

24. W. Nulty and M. Trick, 1988. *GNO/PC Generalized network optimization system,* **Operations Research Letters 7**, 101-102.

25. M. Ramamurti, 1989. *Parallel algorithms for generalized networks*, Ph.D. thesis, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, Texas.

26. J. Tomlin, 1984. *Solving generalized network models in a general purpose mathematical programming system,* **ORSA/TIMS Joint National Meeting** Dallas,TX.

Computational Comparison of Sequential and Parallel Algorithms For the One-to-One Shortest-Path Problem

Richard V. Helgason †
Jeffery L. Kennington †
B. Douglas Stewart ‡

Four new shortest-path algorithms, two sequential and two parallel, for the source to sink shortest-path problem are presented and empirically compared with five algorithms previously discussed in the literature. The new algorithm, S22, combines the highly effective data structure of the S2 algorithm of Dial, Glover, Karney, and Klingman, with the idea of simultaneously building shortest-path trees from both source and sink nodes, and was found to be the fastest sequential shortest-path algorithm. The new parallel algorithm, PS22, is based on S22 and is the best of the parallel algorithms. We also present results for three new S22-type shortest-path heuristics. These heuristics find very good (often optimal) paths much faster than the best shortest-path algorithm.

Since the late fifties when its first solution methods were developed, the shortest-path problem has become one of the fundamental problems in the areas of combinatorial optimization, computer science, and operations research. Algorithms and applications are commonly found in the important books in these areas (see for example Berge and Ghouila-Houri (1962), Bertsekas and Gallager (1987), Even (1979), Hu (1982), Jensen and Barnes (1980), Lawler (1987), Papadimitriou and Steiglitz (1987), Quinn (1984), Rockafellar (1984), and Tarjan (1983)). The study of this problem has been motivated by both its elegant mathematical structure and its many practical applications. Our recent interest in this problem was occasioned by the need to solve shortest-path subproblems in several mathematical optimization procedures we are developing in an MIMD parallel computing environment.

Excellent surveys of the many shortest-path problem variations may be found in Deo and Pang (1984) and Gallo and Pallottino (1986). A survey of techniques and computational comparisons may be found in Gallo and Pallottino (1988), in Dial, Glover, Karney, and Klingman (1977) and (1979), in Klingman, Mote, and Whitman (1978), in Glover, Glover, and Klingman (1984), in Desrochers (1987), and in Divoky (1987). The methods are grouped into two general classes: *label-setting algorithms* and *label-correcting algorithms*. Dijkstra (1959) is credited with the first label-setting algorithm and any algorithm that uses this approach has been considered a particular implementation of Dijkstra's original algorithm (see Gallo and Pallottino (1986)).

Typically, the shortest-path problem is one that requires the shortest-path from a single source node, say s, to all other nodes in a network. The solution can be represented as a shortest-path tree rooted at s. In this paper we are concerned with the problem of finding the shortest-path between a source node and a sink node, t. Dantzig (1960) suggested that a pair of trees be built with one rooted at s and the other rooted at t. No stopping criteria were given. This strategy also appears in the book by Berge and Ghouila-Houri (1962) with an incorrect stopping criterion. Nicholson (1966) was the first to present a correct analysis of the Dijkstra two-tree algorithm. Hart, Nilsson, and Raphael (1968) presented a one-tree algorithm utilizing heuristic

† Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275-0122.

‡ Department of Industrial Engineering, University of Alabama, Tuscaloosa, AL 35406-0288.

This research was supported in part by the Department of Defense under Contract Number MDA 903–86–C0182, the Air Force Office of Scientific Research under Contract Number AFOSR 87–0199, and the Office of Naval Research under Contract Number N00014–87–K–0223.

cost functions. They included the case in which lower bounds on distances between nodes are available and proved that optimal paths are obtained. Pohl (1971) extended these results for the bi-directional algorithm, antedating Mohr and Pasche (1988), who presented similar results. Additional discussion may be found in the survey by Dreyfus (1969), where he conjectured that building trees from s and t would be ineffective for this problem.

Few empirical studies have been reported in the literature for the two-tree algorithms. Pohl (1969) presented results that have received little recognition, and no further studies appear until recently. Helgason, Kennington, and Stewart (1988) solved shortest-path problems on twelve NETGEN networks and twelve dense bipartite graphs using a two-tree Dijkstra algorithm. Mohr and Pasche (1988) solved for shortest-paths in three grid networks and a network representing a road map of Switzerland, using the two-tree Dijkstra algorithm as well as their version of the two-tree algorithm for networks with lower bounds available. They also simulated results for parallel algorithms if two processors were available.

The motivation for this study was to implement two-tree algorithms using more efficient data structures than the classical Dijkstra algorithm, and to actually solve shortest-path problems using two processors. Four new algorithms were developed: sequential and parallel two-tree algorithms based on Dial (1969) (the S1 code in Dial et al. (1979)), and sequential and parallel two-tree algorithms based on the S2 code in Dial et al. (1979). These codes are compared with the classical Dijkstra, S1, S2, two-tree Dijkstra, and parallel two-tree Dijkstra codes.

While performing this study, it was noted that the two-tree S2 algorithm finds "good" paths quickly. Three heuristic path algorithms were developed by simply stopping early while executing the two-tree S2 algorithm. Often times these heuristics find a shortest-path, although they do not prove it is so, and these paths are found much faster than the fastest optimal algorithm. We present computation times as well as measures of closeness to optimality for these heuristics.

1. THE ALGORITHMS

This section presents the definitions, notation, and nine algorithms representing the codes used in our computational study. The notation and presentation are based on that found in Gallo and Pallattino (1986). The input for each algorithm is a directed graph $G = [N,A]$ with a node set N and an arc set A. Associated with each arc $(i,j) \in A$ is a length l_{ij}. A shortest-path is desired between two nodes s and t and it is assumed that such a path exists. We also assume for all algorithms that $l_{ij} \geq 0$ for all $(i,j) \in A$. For efficient implementation it is assumed that G is given in the form of arc-lists. Specifically, the forward star for node $u \in N$, $FS(u)$, is the set of all arcs $(u,j) \in A$. Six algorithms require a backward star, $BS(v)$, defined as the set of all arcs $(i,v) \in A$ for node $v \in N$.

The basic working entities of each algorithm include a set of labels, d_u, for node distances from the root of a shortest-path tree T, a set of predecessors, p_u, for nodes in the tree, and the set of candidate nodes, Q. In the algorithms based on the S1 and S2 algorithms, the set Q will be divided into subsets, or buckets, that will contain candidate nodes that are the same distance from the root node. This requires that each arc length be *integer* to correspond to an index. For ease of presentation, we will let Γ be the set of indices, $\{0, ..., lmax\}$ for the buckets of Q, where $lmax = \max\{l_{uv} : (u,v) \in A\}$. The notation will be modified for the bi-directional algorithms by using the superscripts s or t to indicate the root of the tree. The algorithms terminate with the length of the shortest-path from s to t and a small set of nodes, J, used to identify a shortest-path. A shortest-path from s to t is implicit in the predecessor labels.

1.1 The classical Dijkstra algorithm

Dijkstra's classical algorithm (1959) begins at node s and builds a shortest-path tree T in which the shortest-path from s to any node in the tree is known. When node t is placed in the tree we have a minimum length directed path from s to t and may terminate. As mentioned before, the algorithms discussed here differ in the way the candidate nodes are placed in and retrieved from the set Q. The implementation here for the classical Dijkstra algorithm (D1) searches the list of candidate nodes, Q, for minimum label nodes and places all such nodes in the set R. Then all the nodes in R can be scanned one after the other. When there are many nodes tied with the minimum label, searches are avoided with only minimal effort. The algorithm may be stated as follows:

```
Procedure D1($s,t$)
   begin
      initialize:
         $p_i \leftarrow 0, d_i \leftarrow \infty$ for all $i \in N$; $Q \leftarrow \emptyset$;
         $d_s \leftarrow 0, p_s \leftarrow s, R \leftarrow \{s\}, T \leftarrow \emptyset$;
      while $t \notin R$
         for each $u \in R$ do
            for each $(u,v) \in FS(u)$ such that $d_u + l_{uv} < d_v$ do
               $d_v \leftarrow d_u + l_{uv}$;
               $p_v \leftarrow u$;
               if $v \notin Q$ then $Q \leftarrow Q \cup \{v\}$;
            end
            $T \leftarrow T \cup \{u\}$;
         end
         comment: search $Q$ for minimum label nodes and place in $R$
         $\alpha \leftarrow \min\{d_i : i \in Q\}, R \leftarrow \{i : d_i = \alpha\}, Q \leftarrow Q \backslash R$;
      endwhile
   end
```

1.2 The S1 algorithm

This one-tree algorithm is implemented as proposed by Dial (1969). As in algorithm D1, it selects a minimum label node to scan each iteration, but the nodes in Q are stored in buckets according to their distance labels. More specifically, $lmax + 1$ buckets are required, where $lmax = \max\{l_{uv} : (u,v) \in A\}$ and a node u is stored in bucket z if $d_u = z (\text{mod } lmax + 1)$. Only nodes with equal distance labels will be in bucket z and only $lmax + 1$ buckets are required, because for a node u with minimum label, we have that for each $v \in Q, d_u \le d_v \le d_u + lmax$. The buckets are implemented efficiently as two-way linked-lists. From minimum label node u, the next non-empty bucket contains the next minimum label node(s) and the effort to search an entire list as in algorithm D1 is greatly reduced. The trade-off is increased effort managing the two-way linked-list each time a node has an improved distance label. The algorithm may be stated as follows:

```
Procedure S1($s,t$)
   begin
      initialize:
         $p_i \leftarrow 0, d_i \leftarrow \infty$ for all $i \in N$; $Q_z \leftarrow \emptyset$ for $z = 1, ..., lmax$;
         $Q_0 \leftarrow \{s\}$, $d_s \leftarrow 0, p_s \leftarrow s, T \leftarrow \emptyset$;
      while $t \notin T$
         let $z$ be the next index such that $Q_z \neq \emptyset$ ;
         for each $u \in Q_z$ do
            $Q_z \leftarrow Q_z \backslash \{u\}$;
            for each $(u,v) \in FS(u)$ such that $d_u + l_{uv} < d_v$ do
               $a \leftarrow d_v (\text{mod } lmax + 1)$;
               $d_v \leftarrow d_u + l_{uv}$;
               $b \leftarrow d_v (\text{mod } lmax + 1)$;
               $p_v \leftarrow u$;
               $Q_a \leftarrow Q_a \backslash \{v\}$;
               $Q_b \leftarrow Q_b \cup \{v\}$;
            end
            $T \leftarrow T \cup \{u\}$;
         end
      endwhile
   end
```

1.3 The S2 algorithm

This one-tree algorithm is based on an idea due to Dantzig (1960) and the implementation here is due to Dial et al. (1977). It requires that each $FS(u)$ for all $u \in N$ be sorted in shortest first order. Given this, the observation can be made that the entire forward star of a node u need not be scanned all at once in that the node, say v, that is first updated will have a distance label less than or equal to any subsequent nodes updated from node u. The node v is placed on a one-way linked-list, paired with u, at a level $d_u + l_{uv}$. In stating the algorithm below, we use $k(u)$ as a counter to point to the next arc in $FS(u)$ to be scanned. The node $w_{k(u)}$ is the corresponding node of this arc. The length of each forward arc-list is given by $h(u)$. It should be noted that the actual implementation does not use a counter. A simple minus sign in the forward star array can indicate the next arc to be scanned.

Nodes may be duplicated on the linked-list, therefore no deleting is required. Even so, the number of nodes on the linked-list will not exceed the total number of nodes since only one per scanned node is allowed. Q is searched as in algorithm S1 for the next non-empty bucket and minimum label node. If the node's paired predecessor is not its current predecessor, the node is already in the shortest-path tree and may be discarded. The algorithm may be stated as follows (see Gallo and Pallottino (1986)):

```
Procedure S2($s,t$)
  begin
    initialize:
      $p_i \leftarrow 0, d_i \leftarrow \infty, k(i) \leftarrow 1, h(i) = |FS(i)|$ for all $i \in N$;
      $Q_z \leftarrow \emptyset$ for $z = 1,...,lmax$; $Q_0 \leftarrow \{s\}$, $d_s \leftarrow 0, p_s \leftarrow s$, $T \leftarrow \emptyset$;
    while $t \notin T$
      let $z$ be the next index such that $Q_z \neq \emptyset$;
      for each $u \in Q_z$ do
        comment: determine next node in $FS(u)$
        $v \leftarrow w_{k(u)}$;

        $Q_z \leftarrow Q_z \backslash \{u\}$;
        comment: determine new candidate arc
        INSERT($u$);
        $T \leftarrow T \cup \{u\}$;
        if $v \notin Q$ then INSERT($v$);
      end
    endwhile
  end
Procedure INSERT($x$)
  begin
    $k(x) \leftarrow k(x) + 1$;
    $y \leftarrow w_{k(x)}$;

    while $k(x) \leq h(x)$ and $d_x + l_{xy} \geq d_y$ do
      $k(x) \leftarrow k(x) + 1$;
      $y \leftarrow w_{k(x)}$;
    endwhile
    if $k(x) \leq h(x)$ then
      $p_y \leftarrow x$;
      $d_y \leftarrow d_x + l_{xy}$;
      $a \leftarrow d_y(\text{mod } lmax + 1)$;
      $Q_a \leftarrow Q_a \cup \{x\}$;
    endif
  end
```

1.4 The two-tree Dijkstra algorithm

The two-tree Dijkstra algorithm builds a pair of shortest-path trees, one from s and one from t. The tree rooted at t is analogous to the one rooted at s, but scans backward stars, and its predecessors are the heads of arcs rather than tails. The two trees are grown in alternate steps and termination is triggered when a node appears in both trees. It should be noted however that the node appearing in both trees is not necessarily on a shortest-path (see Nicholson (1966) for a proof of termination criteria). Testing on random graphs showed that about 72% of the time this node will be on the shortest-path. A search is performed over nodes in both trees to find a node, say r, such that $d_r^s + d_r^t$ is a minimum. A shortest-path from s to t may be found by following predecessors from r to s and from r to t in each tree, respectively. We define J as the set of nodes which can be used to identify a shortest-path. The algorithm requires twice the storage of algorithm D1 and may be stated as follows:

```
Procedure D2$(s,t)$
  begin
    initialize:
      $p_i^s \leftarrow 0, d_i^s \leftarrow \infty$ for all $i \in N$; $Q^s \leftarrow \emptyset$;
      $d_s^s \leftarrow 0, p_s^s \leftarrow s, T^s \leftarrow \emptyset, R^s \leftarrow \{s\}$;
      $p_i^t \leftarrow 0, d_i^t \leftarrow \infty$ for all $i \in N$; $Q^t \leftarrow \emptyset$;
      $d_t^t \leftarrow 0, p_t^t \leftarrow t, T^t \leftarrow \emptyset, R^t \leftarrow \{t\}$;
    while $T^s \cap T^t = \emptyset$ do
      for each $u \in R^s$ do
        for each $(u,v) \in FS(u)$ such that $d_u^s + l_{uv} < d_v^s$ do
          $d_v^s \leftarrow d_u^s + l_{uv}$;
          $p_v^s \leftarrow u$;
          if $v \notin Q^s$ then $Q^s \leftarrow Q^s \cup \{v\}$;
        end
        $T^s \leftarrow T^s \cup \{u\}$;
      end
      comment: search $Q^s$ for minimum label nodes and place in $R^s$
      $\alpha \leftarrow \min\{d_i^s : i \in Q^s\}, R^s \leftarrow \{i : d_i^s = \alpha\}, Q^s \leftarrow Q^s \backslash R^s$;
      for each $v \in R^t$ do
        for each $(u,v) \in BS(v)$ such that $d_v^t + l_{uv} < d_u^t$ do
          $d_u^t \leftarrow d_v^t + l_{uv}$;
          $p_u^t \leftarrow v$;
          if $v \notin Q^t$ then $Q^t \leftarrow Q^t \cup \{u\}$;
        end
        $T^t \leftarrow T^t \cup \{v\}$;
      end
      comment: search $Q^t$ for minimum label nodes and place in $R^t$
      $\alpha \leftarrow \min\{d_i^t : i \in Q^t\}, R^t \leftarrow \{i : d_i^t = \alpha\}, Q^t \leftarrow Q^t \backslash R^t$;
    endwhile
    comment: stopping criterion met
    $\beta \leftarrow \min\{d_i^s + d_i^t : i \in T^s \cup T^t\}$;
    $J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
  end
```

1.5 The two-tree S1 algorithm

The two-tree S1 algorithm builds a pair of shortest-path trees, one from s and one from t, using S1 data structures for each tree. As in algorithm D2, termination is triggered when a node is first found to be in both trees. The node r such that $d_r^s + d_r^t$ is minimum gives the shortest-distance from s to t and lies on a shortest-path. The algorithm requires twice the storage of S1 and may be stated as follows:

```
Procedure S12$(s,t)$
  begin
    initialize:
      $p_i^s \leftarrow 0, d_i^s \leftarrow \infty$ for all $i \in N$; $Q_z^s \leftarrow \emptyset$ for $z = 1, ..., lmax$;
      $Q_0^s \leftarrow \{s\}$, $d_s^s \leftarrow 0, p_s^s \leftarrow s$, $T^s \leftarrow \emptyset$;
      $p_i^t \leftarrow 0, d_i^t \leftarrow \infty$ for all $i \in N$; $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$;
      $Q_0^t \leftarrow \{t\}$, $d_t^t \leftarrow 0, p_t^t \leftarrow t$, $T^t \leftarrow \emptyset$;
    while $T^s \cap T^t = \emptyset$ do
      let $z$ be the next index such that $Q^s_z \neq \emptyset$ ;
      for each $u \in Q^s_z$ do
        $Q^s_z \leftarrow Q^s_z \backslash \{u\}$;
        for each $(u,v) \in FS(u)$ such that $d^s_u + l_{uv} < d^s_v$ do
          $a \leftarrow d^s_v (\text{mod } lmax + 1)$;
          $d^s_v \leftarrow d^s_u + l_{uv}$;
          $b \leftarrow d^s_v (\text{mod } lmax + 1)$;
          $p^s_v \leftarrow u$;
          $Q^s_a \leftarrow Q^s_a \backslash \{v\}$;
          $Q^s_b \leftarrow Q^s_b \cup \{v\}$;
        end
        $T^s \leftarrow T^s \cup \{u\}$;
      end
      let $z$ be the next index such that $Q^t_z \neq \emptyset$ ;
      for each $v \in Q^t_z$ do
        $Q^t_z \leftarrow Q^t_z \backslash \{v\}$;
        for each $(u,v) \in BS(v)$ such that $d^t_v + l_{uv} < d^t_u$ do
          $a \leftarrow d^t_u (\text{mod } lmax + 1)$;
          $d^t_u \leftarrow d^t_v + l_{uv}$;
          $b \leftarrow d^t_u (\text{mod } lmax + 1)$;
          $p^t_u \leftarrow v$;
          $Q^t_a \leftarrow Q^t_a \backslash \{u\}$;
          $Q^t_b \leftarrow Q^t_b \cup \{u\}$;
        end
        $T^t \leftarrow T^t \cup \{v\}$;
      end
    endwhile
    comment: stopping criterion met
    $\beta \leftarrow \min\{d^s_i + d^t_i : i \in T^s \cup T^t\}$;
    $J \leftarrow \{i \in T^s \cup T^t : d^s_i + d^t_i = \beta\}$;
  end
```

1.6 The two-tree S2 algorithm

As in the previous two-tree algorithms, the two-tree S2 algorithm uses mirror S2 data structures to build two shortest-path trees. At first one would expect the stopping procedure to be the same as in the previous two-tree algorithms, namely, when a node is in both trees, find the minimum $d^s_i + d^t_i$ for all $i \in T^s \cup T^t$. However, at the time a node is first placed in its second tree, we are not quite ready to search for such a minimum doubly labelled node. In Nicholson (1966) such a node is proven to be on a shortest-path because each node in both trees has had its arc-list fully scanned. In the S2 implementation for each tree this is not the case. In fact, this is the advantage of the one-tree S2 algorithm. In two directions we must perform additional scanning to meet the Nicholson criterion, however, we will not need to manage the linked-lists. All that is needed is to update distance labels and predecessors. Actually we need only scan a subset of arcs from each arc-list. Nicholson proves that any node that is not in either tree when a node is first in both trees will not be on a shortest-path. If arcs were scanned to these nodes, they would still not be in either tree or on a shortest-path, so updating their distance labels would be wasted effort. These nodes also make up the majority of the arc-lists. The only arcs that need to be scanned during the "mop-up" phase are those arcs that have from nodes in one tree and to nodes in the other tree. Since these arcs may be scanned from either node, it is more efficient to consider these arcs from the nodes in the smaller tree. After updating distance labels and predecessors we are ready to search for the minimum doubly labelled node to find our minimum distance from s to t and a shortest-path. The algorithm may be stated as follows:

```
Procedure S22($s,t$)
  begin
    initialize:
      $p_i^s \leftarrow 0, d_i^s \leftarrow \infty, fk(i) \leftarrow 1, fh(i) = |FS(i)|$ for all $i \in N$;
      $Q_z^s \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^s \leftarrow \{s\}$, $d_s^s \leftarrow 0, p_s^s \leftarrow s$, $T^s \leftarrow \emptyset$;
      $p_i^t \leftarrow 0, d_i^t \leftarrow \infty, bk(i) \leftarrow 1, bh(i) = |BS(i)|$ for all $i \in N$;
      $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^t \leftarrow \{t\}$, $d_t^t \leftarrow 0, p_t^t \leftarrow t$, $T^t \leftarrow \emptyset$;
    while $T^s \cap T^t = \emptyset$ do
      let $z$ be the next index such that $Q_z^s \neq \emptyset$ ;
      for each $u \in Q_z^s$ do
        comment: determine next node in $FS(u)$
        $v \leftarrow w_{fk(u)}$;

        $Q_z^s \leftarrow Q_z^s \backslash \{u\}$;
        comment: determine new candidate arc
        SINSERT($u$);
        $T^s \leftarrow T^s \cup \{u\}$;
        if $v \notin Q^s$ then SINSERT($v$);
      end
      let $z$ be the next index such that $Q_z^t \neq \emptyset$ ;
      for each $v \in Q_z^t$ do
        comment: determine next node in $BS(u)$
        $u \leftarrow w_{bk(v)}$;

        $Q_z^t \leftarrow Q_z^t \backslash \{v\}$;
        comment: determine new candidate arc
        TINSERT($v$);
        $T^t \leftarrow T^t \cup \{v\}$;
        if $u \notin Q^t$ then TINSERT($u$);
      end
    endwhile
    comment: mop-up phase from smaller tree
    if $|T^s| < |T^t|$ then
      for each $u \in T^s$ do
        for each $(u,y) \in FS(u)$ do
          if $y \in T^t$ then
            if $d_u^s + l_{uy} < d_y^s$ then
              $d_y^s \leftarrow d_u^s + l_{uy}$;
              $p_y^s \leftarrow u$;
            endif
          endif
        end
      end
    else
      for each $v \in T^t$ do
        for each $(w,v) \in BS(v)$ do
          if $w \in T^s$ then
            if $d_v^t + l_{wv} < d_w^t$ then
              $d_w^t \leftarrow d_v^t + l_{wv}$;
              $p_w^t \leftarrow v$;
            endif
          endif
        end
      end
    endif
```

```
        comment: stopping criterion met
        $\beta \leftarrow \min\{d_i^s + d_i^t : i \in T^s \cup T^t\}$;
        $J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
    end
Procedure SINSERT($x$)
    begin
        $fk(x) \leftarrow fk(x) + 1$;
        $y \leftarrow w_{fk(x)}$;
        while $fk(x) \leq fh(x)$ and $d_x^s + l_{xy} \geq d_y^s$ do
            $fk(x) \leftarrow fk(x) + 1$;
            $y \leftarrow w_{fk(x)}$;
        endwhile
        if $fk(x) \leq fh(x)$ then
            $p_y^s \leftarrow x$;
            $d_y^s \leftarrow d_x^s + l_{xy}$;
            $a \leftarrow d_y^s (\text{mod } lmax + 1)$;
            $Q_a^s \leftarrow Q_a^s \cup \{x\}$;
        endif
    end
Procedure TINSERT($x$)
    begin
        $bk(x) \leftarrow bk(x) + 1$;
        $y \leftarrow w_{bk(x)}$;
        while $bk(x) \leq bh(x)$ and $d_x^t + l_{yx} \geq d_y^t$ do
            $bk(x) \leftarrow bk(x) + 1$;
            $y \leftarrow w_{bk(x)}$;
        endwhile
        if $bk(x) \leq bh(x)$ then
            $p_y^t \leftarrow x$;
            $d_y^t \leftarrow d_x^t + l_{yx}$;
            $a \leftarrow d_y^t (\text{mod } lmax + 1)$;
            $Q_a^t \leftarrow Q_a^t \cup \{x\}$;
        endif
    end
```

1.7 The parallel two-tree Dijkstra algorithm

It is readily observed that in the two-tree shortest-path algorithms, the trees are independent of each other. The only requirement is to check whether or not a node is in the opposite tree. This read-only step causes no interference using multiple processors. This leads to the simplest asynchronous parallel application using two processors, one for each tree. When one processor recognizes that a node is in both trees, it sets a flag to tell the other processor to find the minimum doubly labelled node in its tree while it does the same. The minimum of the two is the minimum path distance from s to t. Again, a shortest-path is implicit in the predecessor labels beginning with the minimum doubly labelled node. In the algorithms below, each processor has its own indentifying number called *procid*. The parallel processing construct called *fork(2)* indicates that two processors are to be used to execute the sections until the *join* construct is reached. The algorithm may be stated as follows:

```
Procedure PD2($s,t$)
  begin
    $flag \leftarrow 0$;
    comment: begin use of two processors
    fork(2)
    if $procid = 1$ then
      initialize:
        $p_i^s \leftarrow 0, d_i^s \leftarrow \infty$ for all $i \in N$; $Q^s \leftarrow \emptyset$;
        $d_s^s \leftarrow 0, p_s^s \leftarrow s, T^s \leftarrow \emptyset, R^s \leftarrow \{s\}$;
      synchronize processors
      while $flag = 0$ do
        for each $u \in R^s$ do
          for each $(u,v) \in FS(u)$ such that $d_u^s + l_{uv} < d_v^s$ do
            $d_v^s \leftarrow d_u^s + l_{uv}$;
            $p_v^s \leftarrow u$;
            if $v \notin Q^s$ then $Q^s \leftarrow Q^s \cup \{v\}$;
          end
          $T^s \leftarrow T^s \cup \{u\}$;
          if $u \in T^t$ then $flag \leftarrow 1$;
        end
        comment: search $Q^s$ for minimum label nodes and place in $R^s$
        $\alpha \leftarrow \min\{d_i^s : i \in Q^s\}, R^s \leftarrow \{i : d_i^s = \alpha\}, Q^s \leftarrow Q^s \backslash R^s$;
      endwhile
      $\beta_s \leftarrow \min\{d_i^s + d_i^t : i \in T^s\}$;
    endif
    if $procid = 2$ then
      initialize:
        $p_i^t \leftarrow 0, d_i^t \leftarrow \infty$ for all $i \in N$; $Q^t \leftarrow \emptyset$;
        $d_t^t \leftarrow 0, p_t^t \leftarrow t, T^t \leftarrow \emptyset, R^t \leftarrow \{t\}$;
      synchronize processors
      while $flag = 0$ do
        for each $v \in R^t$ do
          for each $(u,v) \in BS(v)$ such that $d_v^t + l_{uv} < d_u^t$ do
            $d_u^t \leftarrow d_v^t + l_{uv}$;
            $p_u^t \leftarrow v$;
            if $v \in Q^t$ then $Q^t \leftarrow Q^t \cup \{u\}$;
          end
          $T^t \leftarrow T^t \cup \{v\}$;
          if $v \in T^s$ then $flag \leftarrow 1$;
        end
        comment: search $Q^t$ for minimum label nodes and place in $R^t$
        $\alpha \leftarrow \min\{d_i^t : i \in Q^t\}, R^t \leftarrow \{i : d_i^t = \alpha\}, Q^t \leftarrow Q^t \backslash R^t$;
      endwhile
      $\beta_t \leftarrow \min\{d_i^s + d_i^t : i \in T^t\}$;
    endif
    comment: end use of multiple processors
    join processors
    $\beta \leftarrow \min\{\beta_s, \beta_t\}$;
    $J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
  end
```

1.8 The parallel two-tree S1 algorithm

The parallel two-tree S1 algorithm is similar to the parallel two-tree Dijkstra algorithm. Each processor is assigned one of the nodes, s or t, and builds a tree using the S1 data structure. When a node is found to be in both trees, a flag is set to tell both processors to find the minimum doubly labelled node in its respective tree. The minimum of these two gives the minimum distance path from s to t with a path implicit in the predecessor labels. The algorithm may be stated as follows:

```
Procedure PS12($s,t$)
  begin
    $flag \leftarrow 0$;
    comment: begin use of two processors
    fork(2)
    if $procid = 1$ then
      initialize:
        $p_i^s \leftarrow 0, d_i^s \leftarrow \infty$ for all $i \in N$; $Q_z^s \leftarrow \emptyset$ for $z = 1, ..., lmax$;
        $Q_0^s \leftarrow \{s\}, d_s^s \leftarrow 0, p_s^s \leftarrow s, T^s \leftarrow \emptyset$;
      synchronize processors
      while $flag = 0$ do
        let $z$ be the next index such that $Q_z^s \neq \emptyset$ ;
        for each $u \in Q_z^s$ do
          $Q_z^s \leftarrow Q_z^s \backslash \{u\}$;
          for each $(u,v) \in FS(u)$ such that $d_u^s + l_{uv} < d_v^s$ do
            $a \leftarrow d_v^s (\text{mod } lmax + 1)$;
            $d_v^s \leftarrow d_u^s + l_{uv}$;
            $b \leftarrow d_v^s (\text{mod } lmax + 1)$;
            $p_v^s \leftarrow u$;
            $Q_a^s \leftarrow Q_a^s \backslash \{v\}$;
            $Q_b^s \leftarrow Q_b^s \cup \{v\}$;
          end
          $T^s \leftarrow T^s \cup \{u\}$;
          if $u \in T^t$ then $flag \leftarrow 1$;
        end
      endwhile
      $\beta_s \leftarrow \min\{d_i^s + d_i^t : i \in T^s\}$;
    endif
    if $procid = 2$ then
      initialize:
        $p_i^t \leftarrow 0, d_i^t \leftarrow \infty$ for all $i \in N$; $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$;
        $Q_0^t \leftarrow \{t\}, d_t^t \leftarrow 0, p_t^t \leftarrow t, T^t \leftarrow \emptyset$;
      synchronize processors
      while $flag = 0$ do
        let $z$ be the next index such that $Q_z^t \neq \emptyset$ ;
        for each $v \in Q_z^t$ do
          $Q_z^t \leftarrow Q_z^t \backslash \{v\}$;
          for each $(u,v) \in BS(v)$ such that $d_v^t + l_{uv} < d_u^t$ do
            $a \leftarrow d_u^t (\text{mod } lmax + 1)$;
            $d_u^t \leftarrow d_v^t + l_{uv}$;
            $b \leftarrow d_u^t (\text{mod } lmax + 1)$;
            $p_u^t \leftarrow v$;
            $Q_a^t \leftarrow Q_a^t \backslash \{u\}$;
            $Q_b^t \leftarrow Q_b^t \cup \{u\}$;
          end
          $T^t \leftarrow T^t \cup \{v\}$;
          if $v \in T^s$ then $flag \leftarrow 1$;
```

```
            end
        endwhile
        $\beta_t \leftarrow \min\{d_i^s + d_i^t : i \in T^t\}$;
    endif
    comment: end use of multiple processors
    join processors
    $\beta \leftarrow \min\{\beta_s, \beta_t\}$;
    $J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
end
```

1.9 The parallel two-tree S2 algorithm

As in the previous parallel algorithms, the parallel two-tree S2 algorithm assigns one processor to work on the tree rooted at s and another processor to work on the tree rooted at t. When a node first appears in both trees, a flag is set to initiate the mop-up node scanning phase. As explained in Section 1.6, we perform the mop-up scanning phase from the smaller tree only. To perform this work with two processors requires each one to share the same data, namely, distance labels. To prevent possible interference with each other, parallel processing constructs called *locks* are used so only one processor at a time may update a distance label. Following this, the processors are synchronized and the minimum doubly labelled nodes are found for each tree. The minimum of these two gives the minimum distance path from s to t and a shortest-path is implicit in the predecessor labels. The procedures SINSERT(x) and TINSERT(x) are as presented in Section 1.6. The algorithm may be stated as follows:

```
Procedure PS22($s,t$)
  begin
    $flag \leftarrow 0$;
    comment: begin use of two processors
    fork(2)
    if $procid = 1$ then
      initialize:
        $p_i^s \leftarrow 0, d_i^s \leftarrow \infty, fk(i) \leftarrow 1, fh(i) = |FS(i)|$ for all $i \in N$;
        $Q_z^s \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^s \leftarrow \{s\}$, $d_s^s \leftarrow 0, p_s^s \leftarrow s$, $T^s \leftarrow \emptyset$;
        $p_i^t \leftarrow 0, d_i^t \leftarrow \infty, bk(i) \leftarrow 1, bh(i) = |BS(i)|$ for all $i \in N$;
        $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^t \leftarrow \{t\}$, $d_t^t \leftarrow 0, p_t^t \leftarrow t$, $T^t \leftarrow \emptyset$;
      synchronize processors
      while $flag = 0$ do
        let $z$ be the next index such that $Q_z^s \neq \emptyset$ ;
        for each $u \in Q_z^s$ do
          comment: determine next node in $FS(u)$
          $v \leftarrow w_{fk(u)}$;

          $Q_z^s \leftarrow Q_z^s \setminus \{u\}$;
          comment: determine new candidate arc
          SINSERT($u$);
          $T^s \leftarrow T^s \cup \{u\}$;
          if $v \notin Q^s$ then SINSERT($v$);
          if $u \in T^t$ then $flag \leftarrow 1$;
        end
      endwhile
    endif
```

```
if $procid = 2$ then
   initialize:
      $p_i^t \leftarrow 0, d_i^t \leftarrow \infty, bk(i) \leftarrow 1, bh(i) = |BS(i)|$ for all $i \in N$;
      $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^t \leftarrow \{t\}$, $d_t^t \leftarrow 0, p_t^t \leftarrow t$, $T^t \leftarrow \emptyset$;
      synchronize processors
      while $flag = 0$ do
         let $z$ be the next index such that $Q_z^t \neq \emptyset$ ;
         for each $v \in Q_z^t$ do
         comment: determine next node in $BS(u)$
         $u \leftarrow w_{bk(v)}$;

         $Q_z^t \leftarrow Q_z^t \backslash \{v\}$;
         comment: determine new candidate arc
         TINSERT($v$);
         $T^t \leftarrow T^t \cup \{v\}$;
         if $u \notin Q^t$ then TINSERT($u$);
         if $v \in T^s$ then $flag \leftarrow 1$;
      end
   endwhile
endif
comment: mop-up phase from smaller tree
synchronize processors
if $|T^s| < |T^t|$ then
   comment: each processor works on next unscanned node $u \in T^s$
   for each $u \in T^s$ do
      for each $(u, y) \in FS(u)$ do
         if $y \in T^s \cup T^t$ then
            if $d_u^s + l_{uy} < d_y^s$ then
               set locked
                  $d_y^s \leftarrow d_u^s + l_{uy}$;
                  $p_y^s \leftarrow u$;
               set unlocked
            endif
         endif
      end
   end
else
   for each $v \in T^t$ do
   comment: each processor works on next unscanned node $v \in T^t$
      for each $(w, v) \in BS(v)$ do
         if $w \in T^s \cup T^t$ then
            if $d_v^t + l_{wv} < d_w^t$ then
               set locked
                  $d_w^t \leftarrow d_v^t + l_{wv}$;
                  $p_w^t \leftarrow v$;
               set unlocked
            endif
         endif
      end
   end
endif
synchronize processors
if $procid = 1, \beta_s \leftarrow \min\{d_i^s + d_i^t : i \in T^s\}$;
if $procid = 2, \beta_t \leftarrow \min\{d_i^s + d_i^t : i \in T^t\}$;
join processors
$\beta \leftarrow \min\{\beta_s, \beta_t\}$;
$J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
end
```

2. COMPUTATIONAL EXPERIENCE

All nine algorithms have been coded in FORTRAN and run on a Sequent Symmetry S81 using either one or two Intel 80386 processors. Several factors affect the performance of shortest-path codes. First, the number of nodes is important for Dijkstra-type algorithms, whose majority of work is searching a node-length array for a minimum label node. Second, the average degree ($|A|/|N|$) is important because the Dijkstra-type and S1-type algorithms must scan entire forward (or backward) stars each iteration, while S2-type algorithms typically scan only a subset. Finally, the cost range of a network affects the length and sparseness of the Q array for S1-type and S2-type algorithms, which are searched each iteration. The cost range and degree also affect the number of nodes that tie with a minimum distance label, thereby reducing the number of searches.

Four node levels (1000, 2000, 3000, 4000), ten average degree levels (5, 10, 15, 20, 25, 50, 75, 100, 125, 150), and three cost ranges (1-100, 1-1000, 1-10000) were chosen as being varied enough to demonstrate which factors were influencing performance. The total number of random networks generated was 120. Each code solved twenty problems per network using the same randomly generated s and t nodes, yielding a total of 2400 problems. Each data point in Tables 1–3 is the sum of times (in seconds) to solve the twenty problems. Since the S2-type codes require sorted forward and backward stars, all codes were given sorted forward and backward stars to eliminate this as a relative factor among them. It is debatable whether or not the sorting time should be counted against the S2-type algorithms since it *requires* sorted arc-lists. Here we simply assume that the data is available in pre-sorted order and concede that if it were not available, the S2-type algorithms would not be appropriate.

With nine codes and 120 networks, many comparisons and observations can be made. We highlight the major points of interest. First, there is some overlap with previous studies and we wish to confirm previous results. As in Dial et al. (1979), we find that S1 is better than S2 on only the smallest degree problems. As the forward stars get larger, the savings of scanning only a subset are realized. On four high node number, low degree networks, Mohr and Pasche (1988) found that a D2-type algorithm required about 38% of the time needed by a D1 algorithm. Here we found that the averages for D2 were 65%, 49%, and 23% of that for D1 for the cost ranges 1-100, 1-1000, and 1-10000 respectively.

PS22, the parallel two-tree S2 code, is the overall winner. It had the fastest time on 108 out of the 120 networks, while PS12 was fastest on 12 networks. PS22 was the fastest code when the average degree increased above 10 on networks with 1–100 cost range and when the average degree increased above 5 on networks with 1–1000 cost range. It was always the fastest code when the cost range was 1–10000. PD2 improved on the networks with the lowest number of nodes, lowest cost, and highest degree — all factors that reduce the number of and time for searches for minimum label nodes by increasing ties. Were the degree increased to make these networks much more dense, PD2 might become more competitive.

Overall, S22 was the fastest sequential code, and was even faster than PS12 and PD2. It had the fastest time of the sequential codes on all 120 networks. In general, the time required using two S1-type trees is about 23%, 26%, and 36% of the time using only one S1 tree on the 1-100, 1-1000, and 1-10000 cost ranges respectively. Similarly, two S2-type trees require on average 23%, 27%, and 37% of the time required by the S2 code on 1-100, 1-1000, and 1-10000 cost range networks, respectively.

Dreyfus (1969) comments that savings may accrue for two-tree algorithms if the stopping criterion is reached well before $N/2$ nodes are permanently labelled in each tree. This is definitely the case for random networks. The one-tree algorithms scan approximately 50.4% of the nodes in the network until t is placed in the tree, while the two-tree algorithms scan only about 4.7% of the nodes until one is first placed in both trees. That is, two-tree algorithms scan about 9.3% of the nodes scanned by one-tree algorithms, resulting in the above mentioned savings in time.

It should be noted that we also solved the above problems using the efficient label-correcting code THRESH-X2 (see Glover, Klingman, Phillips, and Schneider (1985)). The total times in seconds for the 1-100, 1-1000, and 1-10000 cost range networks were 877.9, 963.4, and 976.0, respectively. Since this algorithm solves the one-to-all shortest-path problem, it was not included in the tables. We can see, however, that it is more efficient to use label-setting algorithms for the one-to-one problem since they stop before the entire tree is built.

When using two processors, D2 and S12 parallelize nicely as the PD2 and PS12 codes. PD2 averages a speed-up of 1.93 over D2. PS12 averages a speed-up of 1.93 over its sequential counterpart, S12. In fact, on some networks a speed-up over 2.0 is achieved. This is due to ties for the minimum label node. The sequential versions scan all nodes that tie in one tree before moving to the other tree. With two processors, a node that is first scanned from a group with ties could be the one that is placed next in the opposite tree and the remaining tied nodes need not be scanned as they are in the one processor version. Less work in parallel results in a speed-up over 2.0. If the sequential versions scanned a node from each tree alternately, the speed-up would be less than 2.0, but overall this was slightly slower than scanning all nodes that tied.

PS22 averages only a speed-up of 1.40 over S22. This lower speed-up is due to the additional cost of using the parallel processing *locks* during the relatively lengthy mop-up phase. That is, when one processor has *locked* a section of code, the other processor waits idly until the section becomes *unlocked* before it can execute the same section.

3. SHORTEST-PATH HEURISTICS

There are times when it may be of interest to quickly find "good" paths in a network. For example, when finding a starting solution to a network flow problem, paths may be found to send flow from sources to sinks that do not have to be minimum paths. "Good" paths at the start may mean the minimum cost flow will be found more quickly. We find there is a trade-off between the time to find a path and the length of the path relative to a minimum path.

We have seen in Section 2 that the S22 code is the overall fastest sequential code for finding a shortest-path between two nodes. Recall that this algorithm requires a mop-up phase to scan all remaining unscanned arcs in the forward or backward stars of the nodes in each shortest-path tree before a shortest-path can be found. This mop-up phase has been found to take from 3% to 70% of the total time for low degree to high degree networks, respectively. However, before this mop-up phase we have a node that is in both shortest-path trees and a path from s to t implicit in the predecessor labels. It may not be an optimal path, but it is likely to be good and is found much quicker on higher degree networks. The heuristic code H2 used the path implied by this first node in each shortest-path tree.

Following the mop-up phase in the optimal S22 code, there is a search over all nodes in each tree for the one with the minimum sum of its distance labels. This same search may be done at the end of heuristic H2, without doing the mop-up phase, to see if there is a better path than the one implied by the first node in both trees. Heuristic H3 is identical to H2, but performs this additional step.

Paths between s and t are known before a node appears in both shortest-path trees. Heuristic code H1 uses the path implied by the node that first has a finite distance label from both s and t.

Tables 4–6 show the times for twenty problems per network on the same networks used in testing the optimal algorithms. Also shown is the percentage this time was of the optimal S22 algorithm. As expected, substantial savings in time are realized on high degree networks, where the mop-up phase dominates the S22 times. On average, H1 requires about 65% of S22 time, while H2 and H3 require 73% and 75% respectively. However on average, H1 ranges from 81% of the S22 time on the lowest degree networks to 46% of S22 time on the highest degree networks. Similarly, H2 has an average range of 93% to 51% of S22 time and H3 has an average range of 95% to 52% of S22 time.

Tables 7–9 show how good these paths are compared to the actual shortest-paths. On average, H1 found the shortest-path 55% of the time. The average length of its path was 8% greater than the shortest-path and the worst path found averaged 44% greater than the shortest-path. *H2 found 72% of the shortest-*paths (reaffirming the necessity of the mop-up phase). Its average path length was 2.4% greater than the shortest-path and the worst path it found averaged 19% greater than the optimal path. H3 found 93% of the shortest-paths with an average path length 0.5% greater than the shortest-path. Its worst path averaged 5% greater than the shortest-path.

It should be noted that in a few instances, H1 found more of the optimal paths for a given network than H2. (See Table 10, nodes = 1000 and degree = 5.) This is similar to the case in which the first node placed in both shortest-path trees is not necessarily on a shortest-path. Here, the node that first has two finite labels (H1) is on a shortest-path, but is not the node that is first placed in both trees (H2). See Helgason et al. (1988) for an example that demonstrates these cases.

4. CONCLUSION

The objective of this paper has been to present four new shortest-path algorithms, two sequential and two parallel, and to empirically compare them with five algorithms previously discussed in the literature. The new algorithms combine the highly effective data structures of the S1 and S2 algorithms with the idea of building trees from a source node and a sink node in order to find a shortest-path. We found that the new S22 algorithm was the fastest sequential algorithm on all networks. The new parallel algorithm, PS22, was the fastest algorithm on all but the lowest degree networks, where PS12 was the fastest. It appears that the parallel two-tree Dijkstra algorithm, PD2, might be competitive only on very low cost, dense networks.

The secondary topic of this paper is heuristic S22-type algorithms for obtaining near-minimum paths. Three new heuristic shortest-path algorithms were discussed and were shown to find very good (often optimal) paths from a source to a sink much faster than the shortest-path can be found. These heuristics eliminate the time-consuming mop-up phase required in the S22 algorithm and are quite effective on higher degree networks.

5. APPENDIX

Tables 1–3 present the computational results for solving twenty problems for each of the nine shortest-path algorithms discussed above. Tables 4–6 present the computational results for solving twenty problems for the three S2-type heuristics discussed above. Tables 7–9 show how often the heuristics found the optimal solution and how far off the solutions were when they did not.

Table 1. - - Time in seconds for 20 problems (Cost range: 1–100)

		code								
nodes	degree	D1	S1	S2	D2	S12	S22	PD2	PS12	PS22
	5	7.62	0.79	0.84	6.22	0.46	0.46	3.38	0.35	0.41
	10	5.03	1.30	0.94	5.06	0.61	0.54	2.79	0.42	0.43
	15	3.81	1.59	0.90	3.86	0.69	0.48	2.18	0.44	0.42
	20	3.65	2.06	1.25	3.64	0.87	0.55	2.07	0.53	0.46
1000	25	4.17	3.06	1.74	3.52	1.01	0.57	2.00	0.61	0.48
	50	4.03	4.57	2.05	2.50	1.50	0.65	1.49	0.82	0.53
	75	4.67	6.21	2.09	2.27	1.91	0.73	1.36	1.05	0.60
	100	6.65	9.00	4.01	2.50	2.63	0.89	1.36	1.30	0.64
	125	6.74	9.87	4.33	2.10	2.57	0.93	1.25	1.36	0.63
	150	9.46	13.72	5.75	2.53	3.32	1.11	1.40	1.70	0.72
	5	23.02	2.28	2.51	18.50	0.92	0.89	9.61	0.59	0.68
	10	12.88	3.23	2.83	12.98	1.13	0.93	6.80	0.69	0.70
	15	8.73	3.43	2.01	9.67	1.34	0.95	5.27	0.78	0.72
	20	9.63	5.50	3.32	8.88	1.55	0.93	4.82	0.83	0.73
2000	25	9.28	6.55	3.35	8.36	1.89	1.09	4.57	1.06	0.78
	50	10.05	11.14	5.31	5.97	2.87	1.21	3.33	1.52	0.92
	75	9.70	12.56	3.65	4.69	3.22	1.23	2.65	1.59	0.89
	100	13.37	18.91	8.79	4.35	4.00	1.41	2.42	2.18	0.98
	125	17.01	23.21	9.16	4.53	5.18	1.52	2.51	2.45	1.05
	150	18.43	27.33	8.74	4.81	5.91	1.79	2.62	2.89	1.15
	5	35.26	3.23	3.44	31.17	1.30	1.24	15.72	0.79	0.92
	10	20.38	5.00	3.98	19.70	1.55	1.25	10.38	0.90	0.94
	15	17.52	6.86	4.95	17.94	1.96	1.34	9.29	1.09	1.00
	20	14.28	8.14	5.26	13.39	2.22	1.37	7.13	1.21	0.99
3000	25	14.05	10.11	5.96	12.97	2.82	1.47	6.99	1.37	1.08
	50	11.71	12.90	4.56	8.30	3.37	1.53	4.60	1.71	1.11
	75	14.73	18.52	5.32	7.09	4.44	1.73	4.04	2.26	1.23
	100	20.02	27.96	9.09	7.14	5.94	1.96	3.99	3.04	1.28
	125	22.83	33.82	12.85	6.54	6.93	2.14	3.46	3.33	1.33
	150	23.88	34.91	12.52	5.91	6.49	2.14	3.33	3.27	1.32
	5	48.97	4.69	4.83	42.68	1.70	1.58	22.13	0.99	1.17
	10	27.81	6.26	5.42	26.38	2.00	1.59	13.73	1.11	1.18
	15	22.66	8.86	6.05	23.14	2.54	1.72	12.77	1.36	1.27
	20	20.28	11.21	7.03	19.05	2.68	1.71	10.11	1.45	1.28
4000	25	21.14	15.66	9.17	17.76	3.32	1.81	9.39	1.71	1.36
	50	18.55	20.51	8.52	11.65	4.49	1.99	6.46	2.18	1.40
	75	24.84	32.71	14.88	10.82	6.69	2.23	5.78	3.13	1.55
	100	27.68	38.53	13.72	9.12	6.84	2.37	5.03	3.24	1.56
	125	30.99	44.23	13.37	9.17	8.66	2.56	4.93	3.90	1.69
	150	30.05	47.07	10.90	8.34	9.63	2.70	4.44	4.39	1.76
total		655.56	557.49	235.39	425.20	129.15	55.29	227.58	65.59	39.34

Table 2. - - Time in seconds for 20 problems (Cost range: 1–1000)

		code								
nodes	degree	D1	S1	S2	D2	S12	S22	PD2	PS12	PS22
1000	5	30.08	1.16	1.15	8.70	0.78	0.67	4.62	0.51	0.54
	10	20.47	1.52	1.14	9.82	0.87	0.63	5.20	0.55	0.53
	15	20.95	2.28	1.45	9.37	0.98	0.63	4.97	0.62	0.54
	20	15.79	2.31	1.38	8.56	1.11	0.63	4.55	0.66	0.55
	25	14.87	3.11	1.64	7.60	1.15	0.63	4.07	0.69	0.55
	50	10.59	4.77	1.99	6.91	1.83	0.78	3.76	1.04	0.64
	75	10.74	7.26	2.73	7.04	2.50	0.94	3.71	1.26	0.69
	100	10.54	9.24	2.95	6.12	2.91	1.03	3.33	1.54	0.79
	125	11.78	11.68	4.75	6.22	3.58	1.23	3.36	1.80	0.89
	150	9.90	10.46	3.21	5.07	3.31	1.12	2.76	1.72	0.79
2000	5	93.53	2.19	2.33	27.70	1.21	1.06	14.14	0.75	0.80
	10	64.37	3.43	2.52	24.63	1.32	1.02	12.65	0.80	0.79
	15	50.13	4.35	3.12	23.35	1.53	1.06	12.18	0.89	0.80
	20	41.88	5.34	3.31	20.52	1.72	1.02	10.69	0.94	0.79
	25	42.08	7.38	4.78	21.40	2.07	1.11	10.95	1.08	0.84
	50	28.76	12.52	5.96	19.29	3.35	1.33	9.87	1.70	0.99
	75	22.46	14.42	6.12	14.36	3.85	1.40	7.62	1.90	1.03
	100	23.51	19.21	4.62	15.13	5.39	1.74	7.92	2.56	1.22
	125	21.25	20.12	5.61	11.78	5.02	1.68	6.25	2.55	1.14
	150	27.28	28.97	8.25	12.22	6.68	2.00	6.48	3.13	1.29
3000	5	157.41	3.44	3.52	52.47	1.67	1.44	26.95	0.99	1.09
	10	113.51	5.83	4.78	44.69	1.88	1.42	23.37	1.06	1.03
	15	84.83	6.93	4.14	43.59	2.14	1.40	21.86	1.18	1.07
	20	65.96	8.14	4.81	42.00	2.64	1.52	21.12	1.37	1.13
	25	63.72	10.61	5.97	39.18	2.83	1.50	19.74	1.46	1.13
	50	39.78	15.73	7.43	25.20	3.84	1.63	13.08	1.88	1.22
	75	41.44	26.88	11.87	27.83	5.98	2.06	14.06	2.85	1.38
	100	34.30	26.80	9.63	20.02	5.99	1.98	10.48	2.79	1.36
	125	34.88	32.14	13.63	18.56	6.55	2.05	9.72	3.15	1.39
	150	38.17	39.71	13.01	17.74	7.94	2.26	9.30	3.67	1.42
4000	5	269.68	4.98	5.42	78.45	2.03	1.81	39.54	1.17	1.32
	10	172.27	8.38	6.76	81.07	2.51	1.83	40.44	1.37	1.35
	15	105.57	7.47	4.21	52.22	2.40	1.67	26.15	1.33	1.28
	20	100.07	11.86	6.64	62.87	3.17	1.85	31.88	1.64	1.36
	25	81.74	12.56	6.36	56.04	3.36	1.89	28.03	1.76	1.41
	50	54.62	21.43	7.01	40.52	4.94	2.05	20.58	2.39	1.48
	75	51.40	31.54	8.64	36.33	6.52	2.36	18.53	3.26	1.65
	100	51.97	43.10	13.61	34.58	9.74	2.93	17.88	4.75	1.81
	125	39.31	33.45	7.67	26.70	8.98	2.84	13.79	4.43	1.84
	150	51.74	50.83	15.91	24.54	10.33	2.99	12.53	4.82	1.93
total		2223.33	573.53	230.03	1090.39	146.60	61.19	558.11	74.04	42.72

Table 3. - - Time in seconds for 20 problems (Cost range: 1–10000)

		code								
nodes	degree	D1	S1	S2	D2	S12	S22	PD2	PS12	PS22
1000	5	51.21	3.77	3.51	11.88	3.86	2.57	6.24	2.25	1.97
	10	42.60	3.36	2.77	9.03	3.01	1.81	4.80	1.81	1.49
	15	38.74	3.47	2.46	10.66	2.97	1.63	5.62	1.77	1.39
	20	40.55	3.96	2.58	10.70	2.92	1.52	5.62	1.75	1.32
	25	43.22	4.59	2.74	9.99	2.99	1.53	5.30	1.78	1.30
	50	30.62	5.83	2.53	9.21	3.34	1.42	4.88	1.95	1.24
	75	34.97	9.09	4.00	11.77	4.31	1.64	6.16	2.34	1.37
	100	28.98	10.11	3.87	9.32	4.28	1.63	4.93	2.37	1.36
	125	24.48	10.66	3.02	9.21	4.72	1.70	4.87	2.57	1.40
	150	25.63	12.82	3.79	8.10	4.79	1.66	4.33	2.57	1.39
2000	5	170.47	5.02	4.63	30.39	4.32	2.99	15.70	2.47	2.26
	10	152.29	5.32	4.31	25.82	3.57	2.27	13.52	2.11	1.77
	15	150.98	6.23	4.22	31.87	3.70	2.11	16.38	2.13	1.68
	20	129.11	6.69	4.15	33.01	3.88	2.03	16.81	2.21	1.67
	25	109.28	6.65	4.19	26.79	3.73	1.90	13.92	2.15	1.57
	50	101.63	12.15	5.21	28.23	4.87	2.01	14.45	2.64	1.60
	75	90.39	17.34	6.67	28.80	5.86	2.13	14.80	3.15	1.72
	100	66.55	17.62	5.15	27.10	6.51	2.27	14.11	3.43	1.69
	125	62.73	21.42	5.41	28.49	7.54	2.50	14.11	3.86	1.89
	150	65.88	28.84	8.73	27.35	8.46	2.69	14.11	4.32	2.01
3000	5	380.23	6.43	6.07	53.09	4.80	3.44	27.06	2.75	2.54
	10	330.84	7.48	5.82	49.91	4.15	2.67	25.16	2.37	2.05
	15	313.76	9.13	6.07	51.80	4.22	2.53	26.47	2.40	1.95
	20	309.35	11.13	7.24	50.05	4.33	2.37	25.10	2.47	1.89
	25	213.96	10.24	5.24	47.35	4.42	2.26	24.08	2.50	1.86
	50	165.98	16.37	6.01	52.71	6.07	2.43	26.91	3.25	1.93
	75	155.82	26.33	13.15	59.12	8.16	2.82	30.14	4.29	2.12
	100	127.05	29.77	12.26	51.31	8.86	2.92	26.22	4.57	2.16
	125	81.45	24.72	7.66	35.55	7.83	2.54	17.83	4.11	1.90
	150	106.22	41.81	10.07	41.62	9.84	3.00	20.97	5.04	2.15
4000	5	741.87	8.62	8.65	108.49	5.51	4.02	53.39	3.12	2.94
	10	386.83	7.57	5.51	61.50	4.33	2.92	30.72	2.50	2.24
	15	527.01	12.33	8.74	89.12	4.90	2.90	44.47	2.78	2.27
	20	416.12	13.36	7.33	82.19	5.03	2.75	41.75	2.78	2.15
	25	381.66	15.51	8.32	67.48	4.97	2.64	33.89	2.78	2.07
	50	233.50	21.68	6.57	81.93	7.14	2.86	40.43	3.81	2.25
	75	199.77	29.65	8.75	68.66	8.04	2.91	34.36	4.26	2.24
	100	187.13	43.26	19.67	69.30	9.82	3.19	34.60	5.01	2.38
	125	135.70	39.82	11.15	56.98	9.93	3.15	28.98	5.16	2.30
	150	131.56	47.19	14.34	61.04	12.08	3.68	30.88	6.03	2.60
total		6986.12	617.34	262.56	1626.92	224.06	98.01	824.07	121.61	76.08

Table 4. - - Heuristic times in seconds for 20 problems (Cost range: 1–100)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	time	0.33	0.35	0.34	0.34	0.35	0.34	0.34	0.37	0.34	0.34	0.34	
		% of S22	71.2	63.8	70.1	62.2	60.9	52.3	47.5	41.0	36.7	31.0	53.7	
	2000	time	0.69	0.68	0.67	0.66	0.68	0.68	0.66	0.67	0.68	0.67	0.67	
		% of S22	77.3	72.9	70.5	71.6	62.6	56.3	53.4	47.7	44.5	37.5	59.4	0.83
	3000	time	1.03	1.00	0.99	0.98	1.00	0.98	0.98	1.00	1.00	0.98	0.99	61.7
		% of S22	83.0	80.3	74.3	70.9	68.4	64.0	56.5	51.1	47.0	45.9	66.7	
	4000	time	1.33	1.28	1.30	1.31	1.30	1.29	1.30	1.31	1.31	1.34	1.31	
		% of S22	83.9	80.4	75.5	76.7	71.8	65.2	58.4	55.3	51.3	49.6	66.8	
H2	1000	time	0.40	0.42	0.38	0.41	0.41	0.39	0.39	0.41	0.40	0.40	0.40	
		% of S22	86.3	76.4	79.0	74.8	71.4	60.2	53.4	46.2	42.7	36.4	62.7	
	2000	time	0.81	0.77	0.79	0.76	0.80	0.79	0.74	0.75	0.75	0.76	0.77	
		% of S22	90.5	82.9	83.8	81.8	73.1	65.2	60.0	52.9	49.0	42.3	68.2	0.93
	3000	time	1.15	1.12	1.14	1.11	1.14	1.10	1.09	1.11	1.10	1.08	1.11	69.3
		% of S22	92.4	89.4	85.6	80.9	77.5	71.8	63.3	56.6	51.4	50.3	71.9	
	4000	time	1.48	1.44	1.47	1.46	1.48	1.43	1.44	1.43	1.42	1.46	1.45	
		% of S22	93.7	90.7	85.7	85.1	81.5	72.2	64.5	60.4	55.5	54.1	74.3	
H3	1000	time	0.42	0.44	0.40	0.43	0.43	0.41	0.41	0.44	0.41	0.42	0.42	
		% of S22	90.6	80.5	83.2	79.4	74.7	62.7	56.7	48.8	44.7	37.9	65.9	
	2000	time	0.83	0.81	0.83	0.79	0.84	0.81	0.76	0.77	0.77	0.79	0.80	
		% of S22	93.2	86.5	87.3	85.5	76.6	67.2	61.9	54.7	50.8	43.8	70.8	0.97
	3000	time	1.18	1.19	1.18	1.15	1.18	1.17	1.12	1.18	1.12	1.11	1.16	71.9
		% of S22	94.9	95.1	88.0	83.4	80.4	76.3	65.0	60.1	52.5	51.7	74.7	
	4000	time	1.52	1.47	1.51	1.49	1.51	1.47	1.47	1.47	1.46	1.49	1.49	
		% of S22	96.1	92.7	87.7	87.1	83.3	73.9	66.0	61.9	56.9	55.3	76.1	

Table 5.- - Heuristic times in seconds for 20 problems (Cost range: 1-1000)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	time	0.49	0.44	0.43	0.40	0.40	0.42	0.41	0.40	0.41	0.40	0.42	
		% of S22	73.7	70.0	69.0	64.0	62.6	53.5	43.1	38.7	33.2	35.4	54.3	
	2000	time	0.86	0.80	0.75	0.76	0.76	0.77	0.73	0.75	0.72	0.75	0.77	
		% of S22	80.7	78.6	70.9	74.0	68.3	57.9	52.3	43.0	42.7	37.4	60.6	0.92
	3000	time	1.17	1.13	1.08	1.06	1.09	1.06	1.07	1.03	1.06	1.03	1.08	61.4
		% of S22	81.1	79.5	77.2	69.7	72.7	64.9	51.9	51.9	51.5	45.6	64.6	
	4000	time	1.50	1.42	1.39	1.41	1.37	1.38	1.37	1.40	1.37	1.42	1.40	
		% of S22	82.8	77.6	83.5	76.4	72.5	67.3	58.1	47.8	48.0	47.6	66.2	
H2	1000	time	0.59	0.53	0.51	0.49	0.46	0.47	0.48	0.47	0.49	0.45	0.49	
		% of S22	87.7	84.8	81.1	77.2	72.8	60.9	50.9	45.5	40.0	40.5	64.1	
	2000	time	0.98	0.90	0.87	0.85	0.86	0.88	0.84	0.88	0.81	0.85	0.87	
		% of S22	92.7	88.9	82.2	83.3	76.8	66.1	59.7	50.4	48.3	42.6	69.1	1.04
	3000	time	1.34	1.26	1.22	1.23	1.22	1.17	1.23	1.15	1.14	1.14	1.21	70.0
		% of S22	93.1	88.4	87.2	80.9	81.6	71.7	59.5	58.1	55.8	50.6	72.7	
	4000	time	1.68	1.65	1.50	1.58	1.55	1.52	1.53	1.61	1.52	1.57	1.57	
		% of S22	92.8	90.3	89.8	85.4	81.8	74.2	64.7	54.9	53.3	52.5	74.0	
H3	1000	time	0.61	0.56	0.52	0.51	0.48	0.50	0.51	0.49	0.52	0.48	0.52	
		% of S22	91.3	88.7	83.1	80.0	76.1	64.2	53.7	48.0	42.2	42.6	67.0	
	2000	time	1.01	0.97	0.90	0.88	0.88	0.90	0.90	0.91	0.84	0.88	0.91	
		% of S22	95.5	95.9	85.0	85.8	79.0	67.9	64.3	52.4	50.1	44.3	72.0	1.07
	3000	time	1.37	1.29	1.25	1.27	1.26	1.20	1.25	1.18	1.17	1.17	1.24	72.3
		% of S22	95.2	90.4	89.0	83.2	84.1	73.5	60.8	59.3	57.0	51.9	74.4	
	4000	time	1.71	1.69	1.53	1.62	1.58	1.55	1.56	1.64	1.55	1.60	1.60	
		% of S22	94.5	92.3	91.8	87.4	83.6	75.7	66.1	56.1	54.4	53.6	75.6	

Table 6.- - Heuristic times in seconds for 20 problems (Cost range: 1-10000)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	time	2.05	1.54	1.32	1.23	1.21	1.05	1.04	1.03	1.00	1.00	1.25	
		% of S22	79.8	85.2	81.2	80.6	79.2	74.1	63.7	63.3	59.0	60.4	72.6	
	2000	time	2.59	1.86	1.71	1.61	1.53	1.42	1.36	1.34	1.35	1.33	1.61	
		% of S22	86.9	82.0	80.8	79.1	80.5	70.6	64.0	59.0	53.9	49.6	70.6	1.75
	3000	time	2.97	2.26	2.07	1.98	1.81	1.74	1.68	1.69	1.63	1.65	1.95	72.0
		% of S22	86.5	84.5	81.9	83.3	80.1	71.5	59.6	57.8	64.0	54.9	72.4	
	4000	time	3.32	2.53	2.42	2.22	2.19	2.02	1.37	1.98	1.95	2.02	2.20	
		% of S22	82.7	86.6	83.4	80.8	82.9	70.7	58.1	62.1	61.8	54.8	72.4	
H2	1000	time	2.46	1.72	1.51	1.37	1.33	1.15	1.16	1.10	1.07	1.05	1.39	
		% of S22	95.8	95.2	92.6	90.4	86.9	80.6	70.7	67.4	63.3	63.3	80.6	
	2000	time	2.90	2.13	1.94	1.83	1.69	1.55	1.51	1.47	1.48	1.46	1.80	
		% of S22	97.4	93.7	92.0	89.8	88.6	77.3	71.0	65.0	59.0	54.4	78.8	1.94
	3000	time	3.32	2.54	2.30	2.16	2.00	1.91	1.92	1.86	1.72	1.77	2.15	79.8
		% of S22	96.5	94.8	91.2	91.2	88.6	78.7	68.1	63.6	67.8	58.9	79.9	
	4000	time	3.87	2.76	2.68	2.49	2.36	2.24	1.53	2.14	2.08	2.19	2.43	
		% of S22	96.3	94.4	92.4	90.4	89.7	78.3	64.7	67.0	66.1	59.6	79.9	
H3	1000	time	2.48	1.75	1.53	1.40	1.35	1.16	1.19	1.13	1.09	1.08	1.42	
		% of S22	96.7	96.6	94.2	92.0	88.4	82.0	72.6	69.0	64.4	64.7	82.1	
	2000	time	2.94	2.16	1.97	1.86	1.71	1.58	1.59	1.50	1.51	1.50	1.83	
		% of S22	98.6	94.9	93.5	91.4	89.8	78.7	74.6	56.1	60.2	55.6	80.3	1.98
	3000	time	3.35	2.60	2.34	2.20	2.07	1.99	1.96	1.89	1.79	1.84	2.20	81.4
		% of S22	97.5	97.1	92.5	92.6	91.6	81.8	69.6	64.7	70.2	61.2	81.9	
	4000	time	3.91	2.78	2.74	2.52	2.39	2.28	1.56	2.18	2.11	2.22	2.47	
		% of S22	97.3	95.3	94.6	91.7	90.7	79.6	66.1	68.3	67.0	60.5	81.1	

Table 7.- - Solution data for 20 problems (Cost range: 1-100)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	% opt	85.0	55.0	55.0	45.0	70.0	65.0	60.0	65.0	70.0	95.0	66.5	
		%>opt	2.6	11.7	10.9	11.8	6.5	9.4	7.8	4.7	6.0	1.2	7.3	
		worst %	38.6	57.9	48.4	45.9	50.0	42.3	26.7	30.0	37.5	25.0	40.2	
	2000	% opt	50.0	55.0	50.0	50.0	40.0	65.0	60.0	80.0	70.0	40.0	56.0	
		%>opt	9.1	7.0	10.7	10.7	8.4	2.9	7.6	4.2	5.6	12.8	7.9	58.3
		worst %	44.8	30.5	44.6	71.0	50.0	12.5	28.6	26.7	30.0	45.5	38.4	7.6
	3000	% opt	55.0	60.0	40.0	60.0	40.0	65.0	55.0	45.0	70.0	65.0	55.5	39.5
		%>opt	4.6	9.1	9.4	10.3	12.6	9.3	7.2	7.1	4.8	5.6	8.0	
		worst %	29.5	94.5	35.1	53.5	46.8	41.7	25.0	44.4	27.3	22.2	42.0	
	4000	% opt	40.0	70.0	35.0	65.0	50.0	60.0	65.0	65.0	50.0	50.0	55.0	
		%>opt	12.8	3.7	9.8	3.4	6.5	4.8	4.8	5.2	11.0	11.0	7.3	
		worst %	36.2	22.9	46.6	21.7	43.9	20.0	25.0	50.0	50.0	57.1	37.3	
H2	1000	% opt	75.0	85.0	85.0	70.0	80.0	65.0	80.0	95.0	80.0	85.0	80.0	
		%>opt	3.7	1.0	2.2	3.1	1.3	5.0	3.1	0.5	4.8	3.0	2.8	
		worst %	38.6	8.2	26.2	14.3	8.1	23.5	26.7	9.1	33.3	28.6	21.7	
	2000	% opt	65.0	80.0	70.0	80.0	60.0	75.0	85.0	90.0	85.0	75.0	76.5	
		%>opt	2.4	1.7	2.7	1.2	3.1	2.7	1.4	0.8	2.0	4.7	2.3	76.1
		worst %	16.8	22.1	13.8	11.3	16.7	17.6	13.3	12.5	25.0	42.9	19.2	2.6
	3000	% opt	75.0	60.0	65.0	70.0	70.0	60.0	90.0	75.0	85.0	80.0	73.0	19.9
		%>opt	1.8	3.6	2.7	2.9	2.3	5.0	0.7	2.9	1.4	2.8	2.6	
		worst %	16.7	35.0	15.7	19.1	15.2	37.5	6.3	15.4	9.1	25.0	19.5	
	4000	% opt	65.0	55.0	65.0	85.0	70.0	90.0	75.0	75.0	85.0	85.0	75.0	
		%>opt	4.1	4.5	2.6	1.0	2.3	1.0	2.4	2.8	2.4	1.7	2.5	
		worst %	35.5	13.7	20.0	7.0	15.2	12.0	40.0	20.0	16.7	11.1	19.1	
H3	1000	% opt	100.0	90.0	100.0	95.0	100.0	100.0	85.0	100.0	95.0	100.0	96.5	
		%>opt	0.0	0.7	0.0	0.1	0.0	0.0	1.2	0.0	0.6	0.0	0.3	
		worst %	0.0	7.1	0.0	1.8	0.0	0.0	7.7	0.0	10.0	0.0	2.7	
	2000	% opt	100.0	80.0	95.0	100.0	75.0	100.0	90.0	95.0	95.0	90.0	92.0	
		%>opt	0.0	1.7	0.5	0.0	1.8	0.0	1.1	0.4	1.0	1.2	0.8	93.6
		worst %	0.0	22.1	7.2	0.0	12.1	0.0	13.3	5.6	25.0	9.1	9.4	0.6
	3000	% opt	95.0	95.0	95.0	95.0	90.0	85.0	100.0	90.0	90.0	95.0	93.0	7.8
		%>opt	0.1	0.4	0.5	0.7	0.3	2.4	0.0	1.3	1.0	0.6	0.7	
		worst %	2.1	6.5	6.2	8.8	2.0	37.5	0.0	14.3	9.1	9.1	9.6	
	4000	% opt	90.0	95.0	80.0	95.0	100.0	95.0	100.0	95.0	90.0	90.0	93.0	
		%>opt	0.9	0.4	0.9	0.2	0.0	0.7	0.0	0.8	1.4	1.2	0.7	
		worst %	11.2	9.6	9.6	4.3	0.0	12.0	0.0	20.0	18.2	11.1	9.6	

Table 8.- - Solution data for 20 problems (Cost range: 1-1000)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
		% opt	55.0	60.0	35.0	60.0	55.0	50.0	55.0	45.0	40.0	50.0	50.5	
	1000	%>opt	6.6	8.1	12.5	10.9	4.8	11.2	12.8	6.3	12.9	10.4	9.7	
		worst %	25.7	44.7	64.4	165.8	20.3	40.7	129.0	35.2	73.7	70.7	67.0	
		% opt	55.0	70.0	35.0	70.0	50.0	50.0	55.0	45.0	65.0	40.0	53.5	
	2000	%>opt	4.9	5.5	10.0	4.2	5.3	11.9	5.6	8.4	11.3	13.1	8.0	52.6
H1		worst %	19.5	74.1	34.3	31.8	37.4	57.1	23.4	38.2	73.7	82.8	47.2	8.2
		% opt	45.0	55.0	70.0	70.0	50.0	75.0	35.0	75.0	65.0	40.0	58.0	46.9
	3000	%>opt	10.8	3.2	2.7	5.3	8.4	4.3	8.6	4.0	5.4	8.1	6.1	
		worst %	48.4	14.3	20.7	35.4	31.4	23.7	29.1	21.6	46.5	34.3	30.5	
		% opt	25.0	25.0	65.0	45.0	60.0	50.0	60.0	35.0	50.0	70.0	48.5	
	4000	%>opt	10.9	11.3	7.4	10.4	7.8	7.2	6.5	13.1	7.7	5.8	8.8	
		worst %	39.1	36.4	68.6	24.3	53.0	26.9	40.4	38.4	61.3	41.0	42.9	
		% opt	65.0	60.0	65.0	65.0	75.0	70.0	65.0	80.0	85.0	70.0	70.0	
	1000	%>opt	2.8	5.7	4.5	4.9	2.3	1.8	4.1	1.4	0.8	4.9	3.3	
		worst %	38.0	32.9	49.4	49.8	31.5	12.3	12.8	7.7	8.2	31.3	27.4	
		% opt	70.0	65.0	55.0	70.0	80.0	65.0	70.0	60.0	90.0	75.0	70.0	
	2000	%>opt	2.0	1.4	2.3	2.4	0.8	3.4	2.3	3.9	0.4	1.1	2.0	69.9
H2		worst %	14.6	17.4	13.7	15.8	12.5	22.4	17.2	19.1	3.8	7.2	14.4	2.4
		% opt	65.0	65.0	75.0	65.0	75.0	75.0	70.0	70.0	80.0	85.0	72.5	20.7
	3000	%>opt	2.9	1.8	2.6	3.2	1.0	1.7	1.4	2.3	1.5	0.9	1.9	
		worst %	27.3	15.4	19.9	41.7	20.3	12.2	13.0	30.3	11.5	9.3	20.1	
		% opt	65.0	45.0	70.0	60.0	65.0	70.0	80.0	60.0	70.0	85.0	67.0	
	4000	%>opt	1.8	3.7	5.4	2.5	2.3	1.8	0.4	3.7	2.9	0.2	2.5	
		worst %	14.9	25.9	59.2	18.9	29.0	4.8	5.2	26.4	21.3	1.3	20.7	
		% opt	100.0	100.0	95.0	90.0	95.0	85.0	100.0	95.0	95.0	85.0	94.0	
	1000	%>opt	0.0	0.0	0.1	0.5	0.1	0.9	0.0	0.2	0.2	2.8	0.5	
		worst %	0.0	0.0	3.0	8.4	1.2	13.9	0.0	3.9	4.6	31.3	6.6	
		% opt	80.0	95.0	95.0	90.0	95.0	85.0	95.0	95.0	100.0	90.0	92.0	
	2000	%>opt	0.8	0.0	0.2	0.4	0.1	2.5	0.1	0.9	0.0	0.6	0.6	92.1
H3		worst %	8.4	0.6	5.2	4.5	1.3	20.7	1.9	19.1	0.0	7.2	6.9	0.6
		% opt	90.0	95.0	90.0	90.0	95.0	100.0	90.0	95.0	95.0	100.0	94.0	7.6
	3000	%>opt	1.6	0.0	1.5	0.2	0.8	0.0	0.4	0.1	0.2	0.0	0.5	
		worst %	27.3	1.2	19.9	3.1	20.3	0.0	5.0	1.2	5.2	0.0	8.3	
		% opt	80.0	80.0	90.0	90.0	95.0	70.0	95.0	95.0	95.0	95.0	88.5	
	4000	%>opt	0.8	0.7	0.6	0.2	0.0	1.4	0.0	1.0	0.8	0.1	0.6	
		worst %	4.3	8.8	10.4	2.0	0.3	4.8	0.9	21.8	28.9	1.3	8.4	

Table 9.- - Solution data for 20 problems (Cost range:1–10000)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
	1000	% opt	40.0	65.0	45.0	55.0	50.0	50.0	45.0	75.0	45.0	60.0	50.5	
		%>opt	8.4	12.7	9.8	7.3	6.9	14.3	11.8	3.2	7.7	6.8	9.7	
		worst %	32.0	90.7	40.6	35.9	54.9	89.5	48.1	26.2	21.9	59.4	67.0	
	2000	% opt	45.0	55.0	45.0	60.0	40.0	35.0	70.0	65.0	55.0	45.0	53.5	
		%>opt	5.4	6.2	6.3	13.3	13.8	11.6	8.1	6.2	11.2	9.3	8.0	52.6
H1		worst %	20.5	30.8	29.6	57.0	35.1	52.3	55.8	46.0	40.0	18.5	47.2	8.2
	3000	% opt	50.0	70.0	50.0	65.0	45.0	65.0	50.0	40.0	65.0	50.0	58.0	46.9
		%>opt	9.1	5.4	7.5	9.6	12.8	5.7	13.4	12.6	9.0	6.9	6.1	
		worst %	58.8	22.2	52.6	52.0	65.3	42.6	80.0	56.5	99.6	39.0	30.5	
	4000	% opt	40.0	55.0	65.0	50.0	30.0	35.0	60.0	60.0	60.0	60.0	48.5	
		%>opt	8.5	7.1	2.9	8.3	13.8	8.5	6.5	4.9	8.9	5.3	8.8	
		worst %	45.4	35.3	10.5	34.5	32.4	35.4	40.4	20.0	32.3	28.7	42.9	
	1000	% opt	60.0	85.0	55.0	80.0	45.0	80.0	65.0	75.0	75.0	80.0	70.0	
		%>opt	3.3	0.7	2.5	1.1	3.9	1.9	2.3	0.7	2.3	1.3	3.3	
		worst %	12.3	7.4	13.3	11.1	16.2	20.8	27.5	3.5	11.6	11.1	27.4	
	2000	% opt	80.0	75.0	50.0	70.0	80.0	75.0	75.0	65.0	65.0	70.0	70.0	
		%>opt	1.2	1.2	3.0	4.9	3.3	1.5	1.6	3.2	5.6	2.2	2.0	69.9
H2		worst %	12.5	10.0	17.4	30.6	31.8	10.9	15.9	21.6	65.9	23.5	14.4	2.4
	3000	% opt	75.0	75.0	65.0	90.0	80.0	80.0	65.0	70.0	100.0	65.0	72.5	20.7
		%>opt	2.1	2.2	3.5	0.3	1.7	1.4	4.2	2.5	0.0	1.5	1.9	
		worst %	11.7	20.4	21.1	4.4	11.4	10.3	33.1	17.6	0.0	14.3	20.1	
	4000	% opt	60.0	75.0	70.0	65.0	55.0	65.0	80.0	70.0	70.0	70.0	67.0	
		%>opt	1.4	2.5	2.3	2.7	2.8	2.7	0.4	2.4	2.4	2.8	2.5	
		worst %	5.9	27.1	16.3	15.9	9.3	14.7	5.2	19.2	17.5	34.7	20.7	
	1000	% opt	85.0	95.0	85.0	95.0	85.0	85.0	95.0	100.0	95.0	90.0	94.0	
		%>opt	0.8	0.3	0.4	0.2	0.4	0.8	0.1	0.0	0.0	0.2	0.5	
		worst %	6.1	5.2	4.5	2.7	2.4	6.3	1.5	0.0	0.3	2.2	6.6	
	2000	% opt	100.0	90.0	75.0	80.0	90.0	95.0	100.0	95.0	90.0	95.0	92.0	
		%>opt	0.0	0.1	1.2	3.1	0.7	0.4	0.0	0.2	0.4	0.2	0.6	92.1
H3		worst %	0.0	1.6	6.4	18.5	6.5	7.9	0.0	3.0	7.4	2.6	6.9	0.6
	3000	% opt	85.0	100.0	100.0	95.0	95.0	100.0	100.0	95.0	100.0	90.0	94.0	7.6
		%>opt	0.5	0.0	0.0	0.0	0.4	0.0	0.0	0.6	0.0	0.2	0.5	
		worst %	6.8	0.0	0.0	0.6	7.0	0.0	0.0	13.3	0.0	1.7	8.3	
	4000	% opt	95.0	90.0	95.0	85.0	100.0	95.0	95.0	90.0	90.0	95.0	88.5	
		%>opt	0.1	0.0	0.2	1.3	0.0	0.4	0.0	0.9	1.2	0.4	0.6	
		worst %	2.4	0.2	3.4	15.9	0.0	11.4	0.9	12.8	17.5	9.0	8.4	

REFERENCES

C. Berge and A. Ghouila, *Programming, Games, and Transportation Networks*, John Wiley and Sons, Inc., New York, NY (1962).

D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, Englewood Cliffs, NJ (1987).

G. Dantzig [illegible] the Shortest Route Through a Network," *Management Science*, 6 (1960) 187–190.

G. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

N. Deo and C. Pang, "Shortest-Path Algorithms: Taxonomy and Annotation," *Networks*, 14 (1984) 275–323.

M. Desrochers, "A Note on the Partitioning Shortest Path Algorithm," *Operations Research Letters*, 6 (1987) 183–187.

R. Dial, "Algorithm 360: Shortest Path Forest With Topological Ordering," *Communications of the ACM*, 12 (1969) 632–633.

R. Dial, F. Glover, D. Karney, and D. Klingman, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," CCS Report 291, Center for Cybernetic Studies, The University of Texas, Austin, TX 78712 (1977).

R. Dial, F. Glover, D. Karney, and D. Klingman, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," *Networks*, 9 (1979) 215–250.

E. Dijkstra, "A Note on Two Problems in Connexion With Graphs." *Numerische Mathematik*, 1 (1959) 269–271.

J. Divoky, "Improvements for the Thresh X2 Shortest Path Algorithm." *Operations Research Letters*, 6 (1987) 227–232.

S. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," *Operations Research*, 17 (1969) 395–412.

S. Even, *Graph Algorithms*, Computer Science Press, Potomic, MD (1979).

G. Gallo, and S. Pallottino, "Shortest Path Methods: A Unifying Approach," *Mathematical Programming Study*, 26 (1986) 38–64.

G. Gallo, and S. Pallottino, "Shortest Path Algorithms," *Annals of Operations Research*, 13 (1988) 3–79.

F. Glover, R. Glover, and D. Klingman, "Computational Study of an Improved Shortest Path Algorithm," *Networks*, 14 (1984) 25–36.

F. Glover, D. Klingman, N. Phillips, and R. Schneider, "New Polynomial Shortest Path Algorithms and Their Computational Attributes," *Management Science*, 31(1985) 1106–1128.

P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions of Systems Science and Cybernetics*, SSC-4, (1968) 100–107.

R. Helgason, J. Kennington, and D. Stewart, "Dijkstra's Two-Tree Shortest-Path Algorithm," Technical Report 88-OR-13, Department of Operations Research, Southern Methodist University, Dallas, TX 75275 (1988).

T. Hu, *Combinatorial Algorithms*, Addison-Wesley, Reading, MA (1982).

P. Jensen and J. Barnes, *Network Flow Programming*, John Wiley and Sons, Inc., New York, NY (1980).

D. Klingman, J. Mote, and D. Whitman, "Improving Flow Management and Control Via Improving Shortest Path Analysis," CCS Report 322, Center for Cybernetic Studies, The University of Texas, Austin, TX 78712, (1978).

D. Klingman, A. Napier, and J. Stutz, "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimal Cost Flow Network Problems," *Management Science*, 20 (1974) 814–821.

E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, NY (1987).

T. Mohr and C. Pasche, "A Parallel Shortest Path Algorithm," *Computing*, 40 (1988) 281–292.

T. Nicholson, "Finding the Shortest Route Between Two Points in a Network," *The Computer Journal*, 9 (1966) 275–280.

C. Papadimitriou, and K. Steiglits, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ (1987).

I. Pohl, "Bi-directional and Heuristic Search in Path Problems," *SLAC Report No. 104*, Stanford, CA (1969).

I. Pohl, "Bi-directional Search," *Machine Intelligence*, 6, B. Meltzer and D. Michie, eds., Edinburgh University Press, Edinburgh (1971) 127–140.

M. Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, New York, NY (1984).

R. Rockafellar, *Network Flows and Monotropic Optimization*, John Wiley and Sons, Inc., New York, NY (1984).

R. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA (1983).

Technical Report 90-CSE-10

THE SHORTEST AUGMENTING PATH ALGORITHM FOR THE TRANSPORTATION PROBLEM

by

J. Kennington

Z. Wang

Department of Computer Science and Engineering

School of Engineering and Applied Science

Southern Methodist University

Dallas, Texas 75275-0122

Revised February 1991

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

The objective of this study was to develop and empirically test a shortest augmenting path algorithm for the transportation problem. The algorithm maintains dual feasibility and complementary slackness and works toward satisfying primal feasibility. Sophisticated heuristics and a modified scaling method are used to achieve an excellent advanced start. Convergence is assured via the use of the shortest augmenting path procedure using reduced costs for arc lengths. The software implementation of our algorithm is uniformly faster than the other competing software on test problems having a small total supply. As the total supply increases, the algorithm degrades and is slower than the best competing software based on a specialization of the primal network simplex algorithm. This manuscript provides the strongest evidence to-date which indicates that network problems having small total supply can best be solved via dual methods and network problems having large supply can best be solved via primal methods.

ACKNOWLEDGMENT

This research was supported in part by the Department of Defense under Contract Number MDA 903-86-C0182, the Air Force Office of Scientific Research under Contract Numbers AFOSR 87-0199 and AFOSR F49620-89-C-0109, and the Office of Naval Research under Contract Number N00014-89-J-1784.

I. INTRODUCTION

The classical transportation problem (also known as the Hitchcock-Koopmans transportation problem) is to find a least cost set of flows on an uncapacitated bipartite graph. Mathematically, this may be formulated as the following special mathematical program:

$$\text{minimize} \quad \sum_{i,j} c_{ij}\, x_{ij} \tag{1}$$

$$\text{subject to:} \quad \sum_{j} x_{ij} = s_{i}, \quad (i = 1, \ldots, m) \tag{2}$$

$$\sum_{i} x_{ij} = d_{j}, \quad (j = 1, \ldots, n) \tag{3}$$

$$x_{ij} \geq 0, \quad (\text{all } i,\ j) \tag{4}$$

where c_{ij} denotes the unit cost for shipments from source i to demand j, s_i denotes the supply at source i, d_j denotes the demand at destination j, and x_{ij} denotes the flow from source i to destination j. If $\sum s_i = \sum d_j$ then (1) - (4) has a feasible solution; otherwise, it does not. It is well known that if s_i for all i and d_j for all j are integers, then there exists an optimal set of flows which are integral.

The transportation problem plays a very important role in the area of network programming. Not only are there numerous applications of the transportation problem, but every minimal cost network flow problem can be transformed into a transportation problem (see Wagner [1959]). Hence, the algorithm development for the transportation problem may benefit the development of algorithms for the minimum cost flow problem.

The transportation problem is a special case of the minimum cost flow problem and can be solved by any of the network flow algorithms which include the following:

(1) network simplex (Johnson [1966]),

(2) out-of-kilter (Fulkerson [1961]),

(3) relaxation method (Bertsekas and Tseng [1988a]),

(4) interior point algorithm (Karmarkar [1984]), and

(5) shortest augmenting path algorithm (Iri [1960]).

The network simplex algorithm is a specialization of the simplex method for solving network problems. Because of the special structure of the constraint matrix in the network problem, the network simplex algorithm completely eliminates the need for carrying and updating the basis inverse. Furthermore, by using the concept of a rooted basis tree and node potentials, the steps of the simplex algorithm can be performed directly on a network diagram. In addition, special strategies for entering variable selection have been developed which enhances the performance of the algorithm. Computer software implementing these ideas have been successfully used to solve large-scale network problems. Even though the network simplex method does not have a polynomial running time bound, empirical studies have shown that its best computer software implementations are comparable to or better than the other algorithms' implementations for the minimum cost flow problem (Ahuja, Magnanti, and Orlin [1989]). For a complete development of the network simplex algorithm and its computational complexity see Kennington and Helgason [1980], Papadimitriou and Steiglitz [1982], Tarjan [1983], and Ahuja, Magnanti, and Orlin [1989].

The out-of-kilter algorithm is based on the Kuhn-Tucker optimality conditions and is composed of two phases: a primal phase and a dual phase. During the primal phase the dual variables are held fixed and the flows are modified. During the dual phase, the primal variables are held fixed and the duals are modified. The algorithm iterates between the primal and dual phases until the Kuhn-Tucker optimality conditions are satisfied. For a detailed description of the out-of-kilter algorithm, see Kennington and Helgason [1980]. For an extensive computational complexity study of this algorithm, see Ahuja, Magnanti, and Orlin [1989].

The relaxation algorithm (Bertsekas and Tseng [1988a]) is a special implementation of the primal-dual method. It is based on iterative improvement of a Lagrange relaxation functional of the minimum cost flow problem. The complementary slackness conditions are maintained at each iteration while the duals are improved along coordinate ascent directions. Termination occurs when primal feasibility is attained. A software implementation of the algorithm (RELAX) is described in Bertsekas and Tseng [1988b].

The interior point algorithm is a new polynomial-time bound algorithm for solving the linear program. It takes a series of steps though the interior points of the primal feasible region and moves in a decent direction from one interior point to another, eventually converging to a near optimal solution. The algorithm has been implemented by AT&T in a product called the KORBX®† system. For a complete theoretical development of the interior point algorithm, see Karmarkar [1984], Vanderbei, Meketon, and Freedman [1986], Karmarkar, Lagarias, Slutsman, and Wang [1989]. For an empirical study of the algorithm, see Cheng, Houck, Liu, Meketon, Slutsman, Vanderbei, and Wang [1989], and Carolan, Hill, Kennington, Niemi, and Wichmann [1990].

The shortest augmenting path algorithm (SAP) is a dual method. Note that the dual problem of (1) - (4) is

$$\text{maximize} \quad \sum_i s_i \lambda_i + \sum_j d_j \pi_j \tag{5}$$

$$\text{subject to:} \quad c_{ij} - \lambda_i - \pi_j \geq 0, \qquad \text{(all i, j)} \tag{6}$$

where λ_i and π_j are dual variables associated with supply node i and demand node j, respectively. The complementary slackness conditions are

$$(c_{ij} - \lambda_i - \pi_j) x_{ij} = 0, \qquad \text{(all i, j)} \tag{7}$$

† KORBX is a registered trademark of AT&T.

and the value of the expression ($c_{ij} - \lambda_i - \pi_j$) is known as the reduced cost, ω_{ij} , for arc (i, j). The SAP algorithm applied to (1) - (4) maintains a set of duals (λ, π) and a set of flow assignments (x) which satisfy (4), (6), and (7). Systematic changes are made to both the duals and the flow assignments until (2) and (3) are also satisfied. When (2), (3), (4), (6), and (7) are all simultaneously satisfied, the corresponding flow assignments are an optimum for (1) - (4). An extensive theoretical study of the shortest augmenting path algorithm for solving the minimum cost flow problem can be found in Tomizawa [1971], Papadimitriou and Steiglitz [1982], Tarjan [1983], and Ahuja, Magnanti, and Orlin [1989].

In this paper, we present a new algorithm for solving the transportation problem which is a combination of heuristic procedures coupled with SAP. The algorithm consists of five phases: column reduction, reduction transfer, row reduction augmentation, scaling, and shortest path augmentation. The column reduction is a simple heuristic which produces an advanced start. If this heuristic solves the problem, then the algorithm terminates. Otherwise, two sophisticated heuristics are applied (reduction transfer and row reduction augmentation) in an attempt to obtain additional flow assignments. If the advanced heuristics produce an optimal solution, the algorithm terminates. Otherwise, a modified scaling technique is used to obtain additional flow assignments. If the scaling procedure produces an optimal solution, the algorithm terminates. Otherwise, a shortest augmenting path method is used to complete the last flow assignments. The steps of the SAP algorithm can be described as follows:

```
SAP Algorithm for the Transportation Problem
step 1.   Initialization and column reduction heuristic
step 2.   If (2) and (3) are satisfied, then terminate
step 3.   Reduction transfer heuristic
step 4.   Scaling heuristic
```

step 5. If (2) and (3) are satisfied, then terminate

step 6. Row reduction augmentation heuristic

step 7. If (2) and (3) are satisfied, then terminate

step 8. Shortest path augmentation.

This algorithm is a generalization of our previous work on the assignment problem (Kennington and Wang [1989a]) and the semi-assignment problem (Kennington and Wang [1989b]).

II. COLUMN REDUCTION

The column reduction heuristic generates an initial set of flows (x), and duals (λ, π) that satisfy (4), (6), and (7). Note that (6) implies that for each j, $\pi_j \leq c_{ij} - \lambda_i$, for all i. Hence, we begin with $\lambda_i = 0$ for all i, $x_{ij} = 0$ for all i, j and $\pi_j = \min \{ c_{ij} : 1 \leq i \leq m \}$ for all j. For those pairs (i, j) such that $c_{ij} - \lambda_i - \pi_j = 0$, the x_{ij} is set to the largest value possible such that the total flow originating at source i does not exceed s_i and the total flow to destination j does not exceed d_j. Obviously dual feasibility and complementary slackness conditions are preserved by this procedure.

Proposition 1. The time bound for the column reduction is O(mn).

At the termination of the COLUMN REDUCTION procedure (4), (6) and (7) will be satisfied. If (2) and (3) are also satisfied, then an optimum has been found. However, this never occurred in our empirical experiments and additional work was required. In the column reduction, the cost range is a crucial factor which affects the amount of supply that can be assigned to destination nodes. In general, the smaller the cost range, the more flow that can be assigned to destination nodes. One experiment showed that for a cost range of from 0 to 100, almost 75% of the total supply was assigned to destination nodes by the column reduction heuristic; while, for a cost range of from 0 to 100,000, only one-half of the total supply was assigned. Of course, the unassigned supply can be assigned by generating a sequence of shortest augmenting paths from source nodes with unassigned supply to destination nodes with unassigned demand. However, since the shortest path augmentation is computationally expensive (we will discuss this in Section V), there is great motivation to use this heuristic to assign as much flow as possible before using the shortest augmenting path procedure.

In the following two sections, we present two other more sophisticated heuristics: reduction transfer and row reduction augmentation. The application of these two procedures can result in the assignment of additional supply at a very reasonable computational expense.

III. REDUCTION TRANSFER

At the termination of the column reduction procedure, a forest can be formed from the arcs having nonzero flow. A forest composed of four trees (components) is illustrated in Figures 1, 2, and 3. Note that for each tree, at most one node has remaining supply or demand. Since (7) is satisfied, the reduced cost for each arc in each tree is zero.

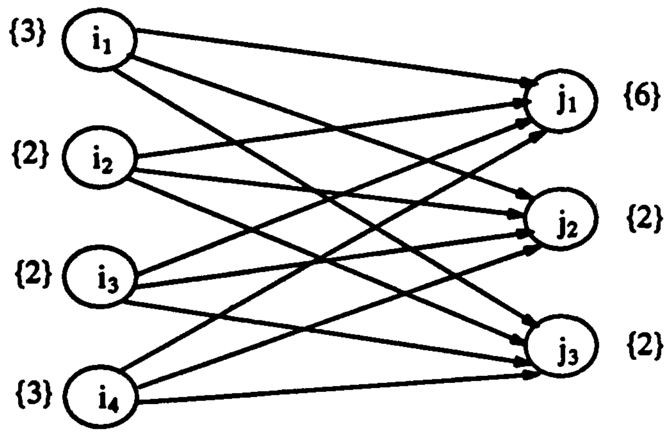

Figure 1. Sample Transportation Problem. ({·} gives the supply (demand) at source (destination) node.)

	j_1	j_2	j_3	
i_1	2	3	3	{3}
i_2	1*	5	2*	{2}
i_3	1*	1*	4	{2}
i_4	7	3	2*	{3}
	{6}	{2}	{2}	

Figure 2. Cost Matrix. (* shows min{c_{ij}} in the column reduction.)

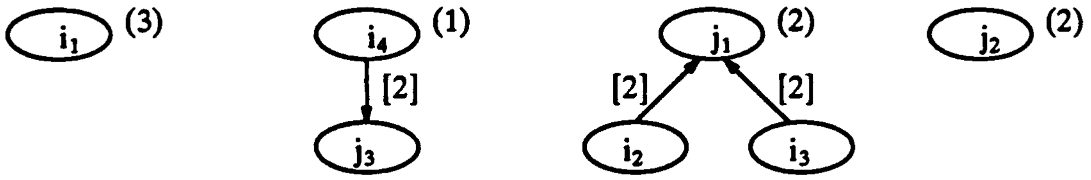

Figure 3. The Forest Associated with the Sample Transportation Problem. ((·) gives the remaining supply (demand) at source (destination) node and [·] denotes the flow on the corresponding arc.)

Let C be a component with remaining supply. Let $I_c = \{i_1, ..., i_s\}$ and $J_c = \{j_1, ..., j_t\}$ be the set of supply and demand nodes in C, respectively. The reduction transfer procedure for the supply component C reduces dual variable π_j of j by some amount $\Delta > 0$, for all $j \in J_c$, and for all $i \in I_c$ increases λ_i by Δ. Dual feasibility considerations determine an upper bound for Δ.

Proposition 2. Let (λ, π, x) denote a set of vectors that satisfy (6) and (7). Let C be a component with a set of supply nodes $I_c = \{i_1, ..., i_s\}$ and a set of demand nodes $J_c = \{j_1, ..., j_t\}$. Select Δ such that $0 \leq \Delta \leq \min_{i \in I_c} [\, \min_{k \notin J_c} (\omega_{ik}) \,]$. Let

$$\hat{\lambda}_i \leftarrow \begin{cases} \lambda_i + \Delta, & i \in I_c \\ \lambda_i, & \text{otherwise} \end{cases}$$

and

$$\hat{\pi}_k \leftarrow \begin{cases} \pi_k - \Delta, & k \in J_c \\ \pi_k, & \text{otherwise.} \end{cases}$$

Then $(\hat{\lambda}, \hat{\pi}, x)$ satisfies (6) and (7).

The proof of Proposition 2 may be found in Wang [1990].

The reduction transfer procedure is presented below:

```
    Procedure REDUCTION TRANSFER
    begin
1.      $I \leftarrow \{1, ..., m\}$, $J \leftarrow \{1, ..., n\}$, and let $\Pi$ denote the set of supply
        components obtained from Procedure COLUMN REDUCTION;
2.      for all $C \in \Pi$
3.          $I_c \leftarrow \{i: i \in C \text{ and } i \in I\}$ and $J_c \leftarrow \{j: j \in C \text{ and } j \in J\}$;
4.          $\Delta \leftarrow \min \{ \omega[i, j]: i \in I_c \text{ and } j \in J - J_c \}$;
5.          $\pi[j] \leftarrow \pi[j] - \Delta$, for all $j \in J_c$;
6.          $\lambda[i] \leftarrow \lambda[i] + \Delta$, for all $i \in I_c$;
7.      end for
    end
```

Note in step 1, we only apply the reduction transfer to supply components which insured that the dual functional will not be reduced.

Proposition 3. The time bound for the reduction transfer procedure is O(mn).

Proof. In the worst case obtaining the final value of Δ takes $O(|I_c||J - J_c|)$ time, and since $\sum |I_c| \leq m$ and $|J - J_c| \leq n$, the time bound for the procedure is $O(mn)$. ■

Even though the reduction transfer does not directly reduce the unassigned supplies, it may help to achieve more flow assignments in the following row reduction augmentation and shortest path augmentation procedures. That is, after the reduction transfer, a saturated demand node j is more expensive to the supply nodes which are not in the same component, and an unexhausted supply node is less expensive to the demand nodes which are not in the same component.

IV. ROW REDUCTION AUGMENTATION

The row reduction augmentation is another heuristic procedure that further reduces the unassigned supply while maintaining dual feasibility as well as the complementary slackness conditions. As we mentioned earlier, the saturated demand nodes are more expensive to unassigned supply nodes which are not in the same component and unassigned supply nodes are less expensive to demand nodes after the reduction transfer; therefore, an application of the row reduction augmentation is more likely to achieve additional flow assignments.

In a classical row reduction, for each unassigned supply node i, we find $\delta = \min \{ \omega_{ik} : k = 1, \ldots, n \}$ and increase λ_i by δ. Then if there exists a demand node j such that $\omega_{ij} = \delta$ and it is unsaturated, we assign f units of flow from source node i to destination node j, where $f = \min \{ s_i - \sum_k x_{ik}, d_j - \sum_l x_{lj} \}$. Dual feasibility and the complementary slackness conditions are maintained and we reduce the total unassigned supply by f.

Proposition 4. Let $I = \{1, \ldots, m\}$, and $J = \{1, \ldots, n\}$, and (λ, π, x) denote a set of vectors that satisfy (6) and (7). Select an $i \in \{l: \sum_k x_{lk} < s_l, l \in I\}$ and let $\delta = \min$

$\{\omega_{ik} : k \in J\}$. Let

$$\hat{\lambda}_l \leftarrow \begin{cases} \lambda_l + \delta, & l = i \\ \lambda_l, & \text{otherwise.} \end{cases}$$

Select a $j \in J$ such that $\omega_{ij} = \delta$.

If $\sum_l x_{lj} < d_j$, then let $f = \min\ \{s_i - \sum_k x_{ik},\ d_j - \sum_l x_{lj}\}$ and

$$\hat{x}_{lk} \leftarrow \begin{cases} x_{lk} + f, & \text{if } l = i \text{ and } k = j \\ x_{lk}, & \text{otherwise.} \end{cases}$$

If $\sum_l x_{lj} = d_j$, select a $p \in I$ such that $p \neq i$ and $x_{pj} \neq 0$, then let

$$f = \min\ \{s_i - \sum_k x_{ik},\ x_{pj}\} \text{ and}$$

$$\hat{x}_{lk} \leftarrow \begin{cases} x_{ij} + f, & \text{if } l = i \text{ and } k = j \\ x_{pj} - f, & \text{if } l = p \text{ and } k = j \\ x_{lk}, & \text{otherwise.} \end{cases}$$

Then $(\hat{\lambda}, \pi, \hat{x})$ satisfies (7).

The proof of Proposition 4 may be found in Wang [1990].

In our study, we modified the classical row reduction procedure by allowing the flow assignment to a saturated demand node to be changed from one source node to another (as described in Proposition 4). Also, we embedded the reduction transfer procedure within the row reduction, so that the deassigned source node may find another unsaturated demand node in the next round of the application of row reduction. We call this modified procedure the row reduction augmentation.

Let rs[i] and rd[j] be the remaining supply and demand at supply node i and demand node j, respectively. The row reduction augmentation procedure may be described as follows:

```
Procedure ROW REDUCTION AUGMENTATION
  begin
1.    $I \leftarrow \{1, \ldots, m\}$ and $J \leftarrow \{1, \ldots, n\}$;
2.    UNEXHAUSTED $\leftarrow \{i : rs[i] > 0, i \in I\}$;
3.    for all $i \in$ UNEXHAUSTED
4.        $\delta \leftarrow \min \{\omega[i, j]: j \in J\}$;
5.        JCAND $\leftarrow \{j : \omega[i, j] = \delta$ and $j \in J\}$;
6.        for all $j \in$ JCAND
7.            if $rd[j] > 0$, then
8.                $f \leftarrow \min \{rs[i], rd[j]\}$
9.                $x[i, j] \leftarrow x[i, j] + f$, $rs[i] \leftarrow rs[i] - f$,
                  and $rd[j] \leftarrow rd[j] - f$;                (flow assignment from i to j)
10.           end if
11.           if $rs[i] = 0$, go to 27;
12.       end for
13.       for all $j \in$ JCAND
14.           $I_j \leftarrow \{l: x[l, j] \neq 0, l \neq i$, and $l \in I\}$;
15.           if $I_j \neq \Phi$, go to 18;
16.       end for
17.       go to 27;
18.       for all $l \in I_j$
19.           $\mu[l] \leftarrow \min \{\omega[l, k]: k \in J, k \neq j\}$;
20.       end for
21.       $\theta \leftarrow \min \{\mu[l]: l \in I_j\}$;
22.       LCAND $\leftarrow \{l: \mu[l] = \theta$ and $l \in I_j\}$;
23.       select a $p \in$ LCAND;
```

```
24.         f ← min {rs[i], x[p, j]}
25.         x[p, j] ← x[p, j] − f and rs[p] ← rs[p] + f;   (deassign f units of flow from p to j)
26.         x[i, j] ← x[i, j] + f and rs[i] ← rs[i] − f;    (assign f units of flow from i to j)
27.         $\lambda[i] \leftarrow \lambda[i] + \delta$;
28.         if rs[i] > 0, then
29.             do steps 4 through 7 of Procedure REDUCTION TRANSFER;
30.         end if
31.     end for
    end
```

Proposition 5. The time bound for the row reduction augmentation is $O(m^2n)$.

Proof. Let $J = \{1, ..., n\}$. For each $i \in$ UNEXHAUSTED, obtaining the value of $\delta = \min \{\omega_{ik} : k \in J\}$ and the index j takes O(n) time. Because $|I_j| \leq m$ and obtaining the value of $\mu[l] = \min \{\omega[l, k]: k \in J, k \neq j\}$ for each $l \in I_j$ takes O(n) time, obtaining the value of θ takes O(mn) time. Therefore, the time bound for the row reduction augmentation on the supply node i is O(mn). Since at most m supply nodes are left unassigned, the total time for the procedure is $O(m^2n)$. ■

The effect of the row reduction augmentation in reducing the total unassigned supply is extraordinary; particularly, for problems with a large cost range. This is due to the fact that the reduction transfer reduces the value of the dual variable π_j associated with the demand node j; so consequently, the source node p which was deassigned from j is likely to be immediately reassigned to another demand node. For a test problem having 400 sources, 400 destinations, total supply of 100,000, and a cost range of [0, 100000], 54,000 units of total supply remained unassigned after the column reduction. A single application of the row reduction augmentation procedure resulted in almost 20,000 units of new flow assignments. A second application of this procedure yielded about 10,000 units of additional flow assignments. For all the problems we tested, the total unassigned supply is reduced to about 20% of the total supply after the procedure is applied four times. We also found that the larger the cost range, the more benefit can be achieved by multiple applications of the procedure. Of course, we can apply the row reduction augmentation procedure as long as the flow is being assigned. However, an inordinate amount of time can be spent in switching supply assigned to a saturated demand node when the procedure tries to complete the last few flow assignments. Also, this heuristic may never achieve a complete flow assignment. Hence, the row reduction procedure should be terminated after a given number of applications. Based on our experience, we recommend application of the row reduction augmentation procedure T times where $T = \max\{ 1, \mathrm{INT}(\log_{10} [\max (c_{ij} : \text{for all } i, j)] - 1) \}$ and $\mathrm{INT}(\alpha)$ denotes the largest integer such that $\mathrm{INT}(\alpha) \leq \alpha$.

V. SHORTEST AUGMENTING PATH

The shortest augmenting path procedure is used to complete the last few flow assignments. In each iteration, this procedure selects an unexhausted supply node, say i, and builds a shortest path from i to an unsaturated demand node using the reduced cost for arc lengths. Then, it augments some amount of flow f along this path by assigning f additional units of flow from all supply nodes on the path to their succeeding demand nodes, and deassigning f units of flow from all supply nodes to their proceeding demand nodes. The total flow is increased by f with each application of this procedure. We use a label setting algorithm to obtain the shortest path and stop the procedure whenever an unsaturated demand node has been permanently labeled.

Proposition 6. The shortest augmenting path procedure has a time bound of $O(Umn)$, where U is the total supply.

The proof of Proposition 6 and a detailed description of our implementation of the label setting algorithm and dual updating procedure may be found in Wang [1990].

From Proposition 6, we can see that using the shortest augmenting path (SAP) procedure to solve the transportation problem may be very expensive when the total supply is large. For example, let i be an unexhausted supply node with 4 units of remaining supply. In the worst case, it probably requires the development of four shortest path trees in order to complete these 4 units of flow assignments. Consider the shortest path tree illustrated in Figure 4. Supply node i has 4 units of remaining supply and demand node j has 3 units of unsatisfied demand. However we can not send 3 units of flow through the path $\{i, j_2, i_2, j\}$ because the residual capacity of the arc (j_2, i_2) is 1. Suppose f is the amount of flow which can be sent along the shortest augmenting path. Clearly, $f \leq \min\{s_i - \sum x_{ik},\ d_j - \sum x_{lj}\}$. If $f = s_i - \sum x_{ik}$, then supply node i is exhausted and one will select another unexhausted supply node, if any, to continue the work. If $f < s_i - \sum x_{ik}$, then we can reuse part of the shortest augmenting path tree. The following proposition presents the theory we use in saving part of the shortest path tree.

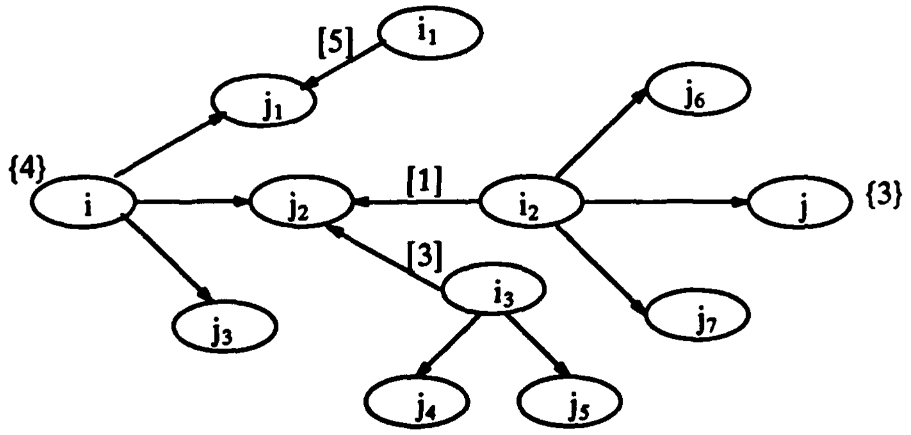

Figure 4. Sample Shortest Path Tree
({·} gives the remaining supply(demand) at root (tail) node and [·] denotes the flow or residual capacity on the corresponding arc.)

Proposition 7. Let $I = \{1, ..., m\}$, $J = \{1, ..., n\}$, and (λ, π, x) denote a set of vectors that satisfy (6) and (7). Select an $i \in \{ l: \sum_k x_{lk} < s[l],\ l \in I\}$ and let q be an unsaturated demand node. Let $S = \{ i, j_1, i_1, \ldots, j_t, i_t, q \}$ be the shortest alternating path from i to q. Suppose that dist[j] is the distance label of j when the procedure is terminated; P is the set of demand nodes with permanent distance labels; and Q is the set of supply nodes whose predecessors are in P. Let δ be the current minimum distance label of demand nodes in the set T, and let f be the maximum flow that can be augmented along the shortest path S, i.e. $f = \min \{ s_i - \sum x_{ik},\ d_j - \sum x_{ij},\ x_{i_1 j_1}, \ldots, x_{i_t j_t} \}$. Let jpred[j] be the predecessor of j, $j \in J$; and ipred[i] be the predecessor of i, $i \in I$. Let

$$\hat{\lambda}_l \leftarrow \begin{cases} \lambda_l + \delta, & l = i \\ \lambda_l - \text{dist}[\text{ipred}[l]] + \delta, & l \in Q \setminus \{i\} \\ \lambda_l, & \text{otherwise,} \end{cases}$$

$$\hat{\pi}_k \leftarrow \begin{cases} \pi_k + \text{dist}[k] - \delta, & k \in P \\ \pi_k, & \text{otherwise.} \end{cases}$$

If $f < s_i - \sum x_{ik}$ and $f < \min \{ x_{i_1 j_1}, \ldots, x_{i_t j_t} \}$, then let

$$\hat{\text{dist}}[k] \leftarrow \begin{cases} 0, & k \in P \\ \text{dist}[k], & \text{otherwise,} \end{cases}$$

$\hat{P} \leftarrow P$, $\hat{T} \leftarrow T$, $\hat{Q} \leftarrow Q$, $\hat{\text{jpred}} \leftarrow \text{jpred}$, $\hat{\text{ipred}} \leftarrow \text{ipred}$, and $\hat{\delta} \leftarrow 0$.

If $f = \min \{ x_{i_1 j_1}, \ldots, x_{i_t j_t} \}$ and $f < s_i - \sum x_{jk}$, then let

$h = \min \{ r: x_{i_r j_r} = f \text{ and } r = 1, \ldots, t \}$,

$P_h \leftarrow \{ k : k \in P$ and k became permanently labeled before $j_h \}$, and

$Q_h \leftarrow \{ l : l = i \text{ or } l \in Q \text{ and } \text{ipred}[l] \in P_h \}$. Let

$$\hat{\text{ipred}}[l] \leftarrow \begin{cases} \text{ipred}[l], & l \in Q_h \\ 0, & \text{otherwise,} \end{cases}$$

$$\hat{\text{jpred}}[k] \leftarrow \begin{cases} \text{jpred}[k], & k \in P_h \\ \text{jpred}[k], & k \notin P_h \text{ and } \text{jpred}[k] \in Q_h \\ l, \text{ where } l \in \{ \hat{\omega}_{lk} = \hat{\text{dist}}[k],\ l \in Q_h \}, & k \notin P_h \text{ and } \text{jpred}[k] \notin Q_h, \end{cases}$$

$$\hat{\text{dist}}[k] \leftarrow \begin{cases} 0, & k \in P_h \\ \text{dist}[k] - \delta, & k \notin P_h \text{ and } \text{jpred}[k] \in Q_h \\ \min\{\hat{\omega}_{lk} : l \in Q_h\}, & k \notin P_h \text{ and } \text{jpred}[k] \notin Q_h, \end{cases}$$

and $\hat{P} \leftarrow P_h$, $\hat{T} \leftarrow \{ k : \hat{\text{dist}}[k] = 0,\ k \in J \setminus P_h \}$, $\hat{Q} \leftarrow Q_h$, and $\hat{\delta} \leftarrow 0$.

Then $\hat{\text{dist}}$ is a set of valid distance labels, i.e., $\hat{P}$ is the set of demand nodes with $\hat{\text{dist}}$ as new permanent distance labels; $\hat{Q}$ is the set of supply nodes whose predecessors are in $\hat{P}$ with the root node i; $\hat{\delta}$ is the current minimum distance label of demand nodes in the set $\hat{T}$; $\hat{\text{jpred}}[j]$ is the predecessor of j; and $\hat{\text{ipred}}[i]$ is the predecessor of i.

A detailed proof of Proposition 7 may be found in Wang [1990].

We have implemented Propositions 7 in our algorithm, so that the procedure will continue to use the shortest augmenting path spanning tree rooted at supply node i until its supply is exhausted. The empirical analysis showed that by doing so it saves about one-third of the time required for this procedure.

VI. THE ALGORITHM

The shortest augmenting path algorithm for the transportation problem can be stated as follows:

```
Procedure SAP_T
  begin
1.  Perform a COLUMN REDUCTION as described in Section II;
2.  If an optimum has been obtained, then terminate;
3.  Perform a REDUCTION TRANSFER as described in Section III;
4.  Perform a ROW REDUCTION AUGMENTATION as described in Section IV;
5.  If an optimum has been obtained, then terminate;
6.  Perform the SHORTEST AUGMENTING PATH Procedure as described in
    Section V until an optimum has been obtained.
7. end
```

Theorem 1. Let U be the total supply and suppose $U \geq \max\{m, n\}$. Then the shortest augmenting path algorithm (Procedure SAP_T) will achieve an optimal solution in O(Umn) time.

Proof. From Propositions 1, 3, 5, and 6, the column reduction takes O(mn) time, the reduction transfer takes O(mn) time, the row reduction augmentation takes $O(m^2 n)$, and the shortest path augmentation takes O(Umn). Since $U \geq m$, SAP_T has a time bound of O(Umn). ■

This algorithm has been implemented in a FORTRAN code called SAP_T. It stores the costs in a single dimensioned array of length mn. In addition it uses six m length arrays, eight n length arrays, and four arrays of length max{m, n}.

The codes which we obtained for comparison are RELAX (Bertsekas and Tseng [1988a and 1988b]), NETFLO (Kennington and Helgason [1980]), and NETSTAR developed by Richard S. Barr of Southern Methodist University. RELAX is a special implementation of the primal-dual algorithm. Both NETFLO and NETSTAR are special implementations of the network simplex method. Kennington and Helgason have compared their code NETFLO with RNET which is a network simplex code developed by T. Hsu and

M. Grigoriadis at Rutgers University and the empirical results showed that NETFLO is about 4% slower than RNET on comparable problems. We believe that NETSTAR is one of the world's fastest pure network codes.

Tables 1 through 5 present our empirical results with RELAX, NETFLO, and NETSTAR on eighty randomly generated problems. The total supply varies from 1000 to 100,000. The column entitled size gives the number of sources and number of destinations so that the number of nodes is always double the size. The problems were all dense so that the number of arcs is the size squared. None of these codes exploited the fact that the problems were dense. The test runs were performed on a Sequent Symmetry S81 at SMU. This machine is configured with 32 Mbytes of memory and runs a modified version of UNIX™†. A summary of Tables 1 through 5 is presented in Figure 5. NETSTAR is clearly the fastest of the three codes. Both NETSTAR and NETFLO are very robust over a wide range of parameters. RELAX is very sensitive to the total supply and required more than twice the time to solve these eighty test problems as did NETSTAR.

† UNIX is a trade mark of AT&T Bell Laboratories.

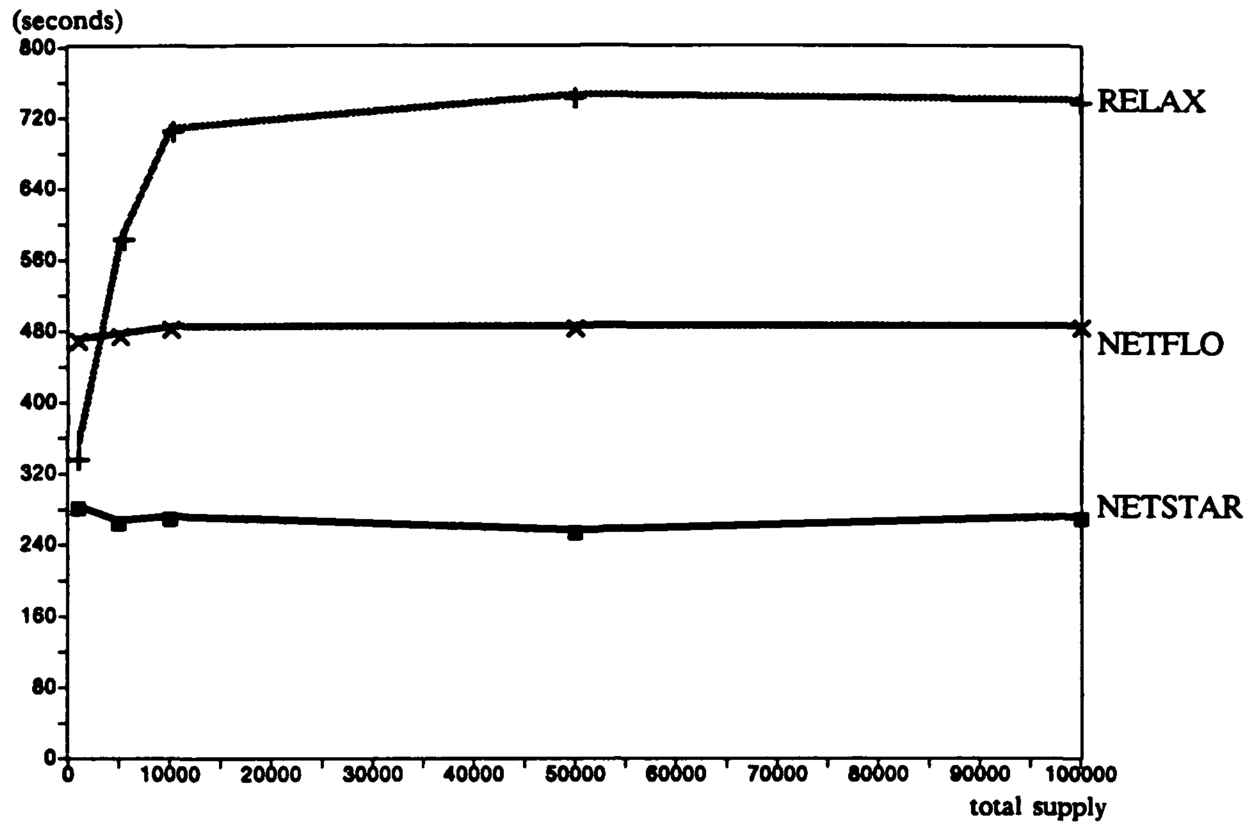

Figure 5. Performance of RELAX, NETFLO, and NETSTAR.
Each time is a sum of solution times of 16 mxm transportation problems, in which m ranges from 100 to 400, maximum cost ranges from 100 to 100,000.
Total solution times for 80 problems:
RELAX = 3150.18 (secs.), NETFLO = 2418.69 (secs.), NETSTAR = 1362.03 (secs.).

Tables 6 through 10 present our empirical results comparing NETSTAR with SAP_T. The shortest augmenting path code is very sensitive to the total supply and was dominated by NETSTAR on four of the five test sets. Where as NETSTAR requires approximately the same time for each of the five test sets (we attribute this fact to the unique feature of the primal simplex method which starts with a feasible solution in the process of finding an optimum), SAP_T requires 3.5 times as much time to solve the fifth test set as the first. This is consistent with the complexity result presented in Theorem 1. The performance of SAP_T and NETSTAR as a function of total supply is illustrated in Figure 6.

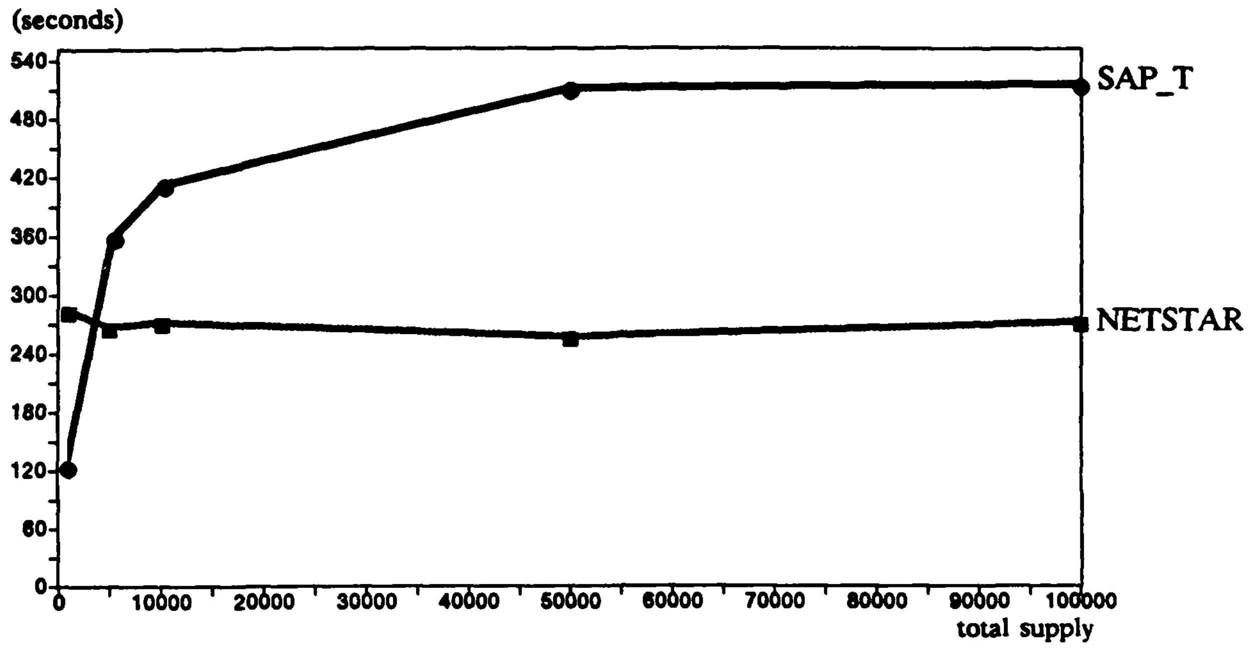

Figure 6. Performance of SAP_T and NETSTAR as a function of total supply. Each time is a sum of solution times for 16 mxm transportation problems, in which m ranges from 100 to 400, maximum costs range from 100 to 100,000.

VII. THE SCALING TECHNIQUE

One technique which can minimize the effect of the total supply on the performance of SAP_T is the scaling technique. The idea is to modify the shortest augmenting path procedure so that larger augmentations are possible during the early stages of the algorithm. Hopefully, fewer augmentations will be required to attain optimality. The scaling technique we use is called the Edmonds-Karp scaling method (Edmonds and Karp [1972]) or right-hand-side scaling (Orlin [1988]).

The shortest augmenting path algorithm with the scaling feature for solving the transportation problem has been implemented in a FORTRAN code called SAP_S_T and compared with NETSTAR. The empirical analysis is presented in Tables 11 through 15. SAP_S_T was some 35% faster than SAP_T but was still sensitive to the total supply. For the test runs with total supply of 1,000 and 5,000, SAP_S_T was faster than NETSTAR. For the test runs with total supply of 10,000, 50,000, and 100,000, NETSTAR dominated SAP_S_T. A summary of all the empirical results with all codes is illustrated in Figure 7.

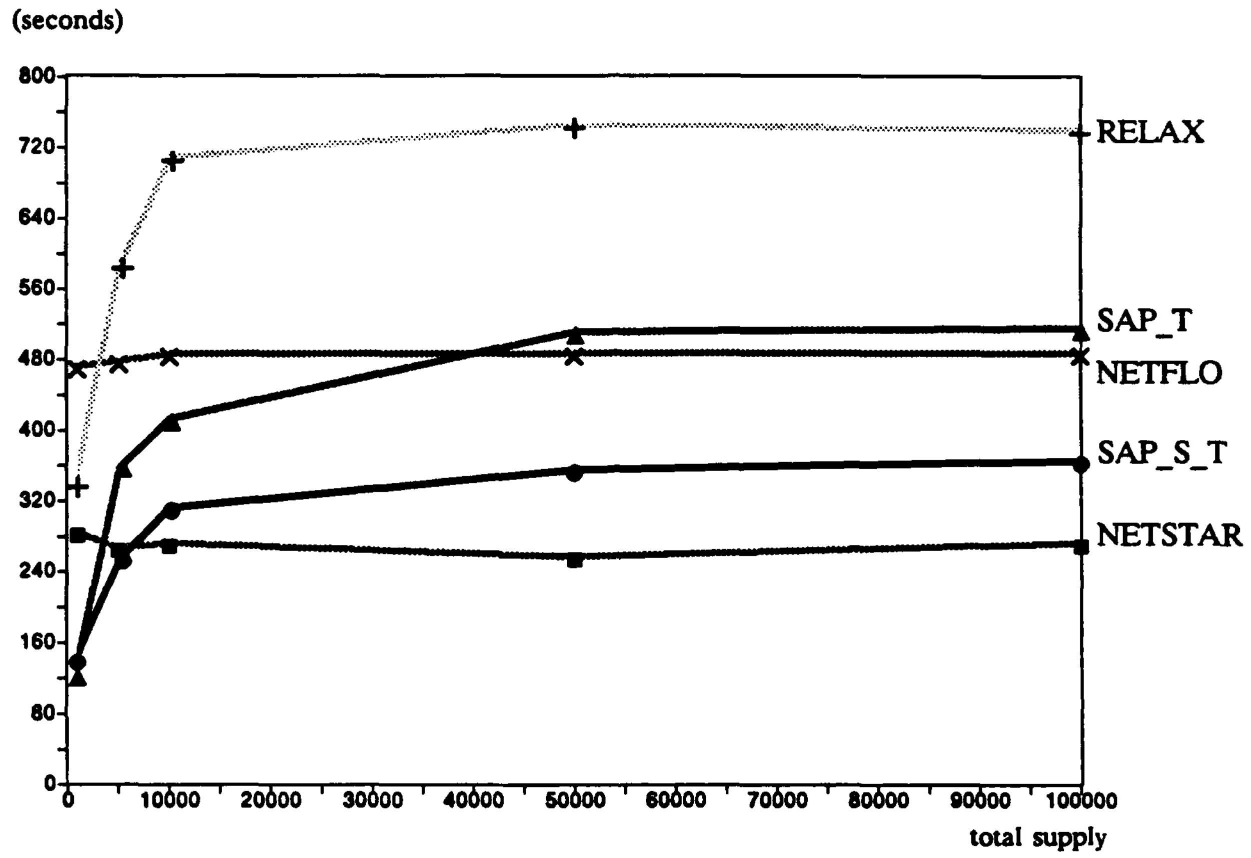

Figure 7. Performance of RELAX, NETFLO, NETSTAR, SAP_T, and SAP_S_T. Each time is a sum of solution times of 16 mxm transportation problems, in which m ranges from 100 to 400, maximum cost ranges from 100 to 100,000. Total solution times for 80 problems:
RELAX = 3150.18 (secs.), NETFLO = 2418.69 (secs.), NETSTAR = 1362.03 (secs.), SAP_T = 1950.06 (secs.), SAP_S_T = 1443.54 (secs.).

SAP_S_T dominates both RELAX and NETFLO on all test problem sets. For small amounts of total supply (below 5,000) SAP_S_T is faster than NETSTAR. When the total supply was at least 10,000, NETSTAR dominated all of the codes.

VIII. SUMMARY AND CONCLUSIONS

In this study, we present an extension of the shortest augmenting path algorithm to the transportation problem. This approach has been extremely successful when applied to both the assignment and the semi-assignment problem dominating all other competing algorithms by a wide margin. In this study, we found that this algorithm is extremely sensitive to the total supply and degrades as the total supply increases. For problems with small total supply, the algorithm is the best method available outperforming the highly successful NETSTAR. The scaling method is very effective when applied to this algorithm reducing the total time by approximately 35%. The key to success with this type of algorithm is the use of clever heuristics to get an advanced start prior to application of methods for generating shortest path trees.

Both empirical evidence and the complexity analysis confirm our belief that network problems which have small total supply can best be solved using dual methods and network problems which have large total supply can best be solved using primal methods. We believe that the present study is the first to clearly demonstrate this phenomena. This conclusion is consistent with our previous studies on the assignment problem and the semi-assignment problem.

REFERENCES

Ahuja, R., T. Magnanti, and J. Orlin, [1988], "Network Flows," Handbooks in Operations Research and Management Science Volume 1: Optimization, Editors G. Nemhauser, A. Rinnooy Kan, and M. Todd, North-Holland, Amsterdam, 211–369.

Bertsekas, D. and P. Tseng, [1988a], "Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems," Operations Research, 36, 93–114.

Bertsekas, D. and P. Tseng, [1988b], "The Relax Codes for Linear Minimum Cost Network Flow Problems," Annals of Operations Research, 13, 125–190.

Carolan, W., J. Hill, J. Kennington, S. Niemi, and S. Wichmann, [1990], "An Empirical Evaluation of the KORBX® Algorithms for Military Airlift Applications," Operations Research, 38, 240–248.

Cheng, Y., D. Houck, J. Liu, M. Meketon, L. Slutsman, R. Vanderbei, and P. Wang, [1989], "The AT&T KORBX® System," AT&T Technical Journal, 68, 3, 7–19.

Edmonds, J. and R. Karp, [1972], "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," Journal of ACM, 19, 248–264.

Fulkerson, D., [1961], "An Out-of-Kilter Method for Minimal Cost Flow Problems," SIAM Journal of Applied Mathematics, 9, 18–27.

Iri, M., [1960], Network Flows, Transportation and Scheduling, Academic Press, New York, NY.

Johnson, E., [1966], "Networks and Basic Solutions," Operations Research, 14, 619–624.

Jonker, R. and T. Volgenant, [1987], "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems," Computing, 38, 325–340.

Karmarkar, N., [1984], "A New Polynomial Time Algorithm for Linear Programming," Combinatorica, 4, 373–395.

Karmarkar, N., J. Lagarias, L. Slutsman, and P. Wang, [1989], "Power Series Variants of Karmarkar-Type Algorithms," AT&T Technical Journal, 68, 3, 20-36.

Kennington, J. and R. Helgason, [1980], Algorithms for Network Programming, John Wiley and Sons, New York, NY.

Kennington, J. and Z. Wang, [1989a], "An Empirical Analysis of the Dense Assignment Problem," Technical Report 88-OR-16, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, TX 75275.

Kennington, J. and Z. Wang, [1989b], "A Shortest Augmenting Path Algorithm for the Semi-Assignment Problem," to appear in Operations Research.

Orlin, J., [1988], "A Faster Polynomial Minimal Cost Flow Algorithm," Proc. 20th ACM Symp. on the Theory of Comp., 377-387.

Papadimitriou, C. and K. Steiglitz, [1982], Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ.

Tarjan, R., [1983], Data Structures and Network Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA.

Tomizawa, N., [1971], "On Some Techniques Useful for the Solution of Transportation Network Problems," Networks, 1, 173-194.

Vanderbei, R., M. Meketon, and B. Freedman, [1986], "A Modification of Karmarkar's Algorithm for Linear Programming," Algorithmica, 1, 395-407.

Wagner, H., [1959], "On a Class of Capacitated Transportation Problems," Management Science, 5, 304-318.

Wang, Z., [1990], "The Shortest Augmenting Path Algorithm for Bipartite Network Problems," unpublished dissertation, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275-0122.

Table 1. Comparison of RELAX, NETFLO, and NETSTAR on nxn Transportation Problems Having 1,000 Units of Supply

Problem Description				RELAX		NETFLO		NETSTAR	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Iterations	Time (secs.)	Iterations	Time (secs.)
100	200	10000	0 – 100	8	1.54	820	2.61	515	1.47
			0 – 1000	9	2.12	1000	2.97	603	1.66
			0 – 10000	10	3.06	954	2.98	575	1.38
			0 – 100000	10	2.88	954	2.94	582	1.60
200	400	40000	0 – 100	6	5.90	2064	10.90	1601	8.50
			0 – 1000	9	9.54	2435	13.73	1841	8.35
			0 – 10000	11	25.48	2543	13.76	1836	9.45
			0 – 100000	13	44.69	2445	13.07	1855	8.82
300	600	90000	0 – 100	7	14.29	3829	30.23	2629	18.74
			0 – 1000	8	19.05	4539	32.14	2789	19.90
			0 – 10000	14	42.47	4556	37.89	2709	21.04
			0 – 100000	14	39.16	4872	35.34	3184	23.28
400	800	160000	0 – 100	6	24.43	5675	62.21	3021	32.18
			0 – 1000	7	31.24	6911	66.89	4784	44.61
			0 – 10000	12	49.19	6901	66.42	4615	43.16
			0 – 100000	14	49.02	7811	79.87	4580	42.30
TOTAL				158	364.06	58309	473.95	37719	286.30

Table 2. Comparison of RELAX, NETFLO, and NETSTAR on nxn Transportation Problems Having 5,000 Units of Supply

Problem Description				RELAX		NETFLO		NETSTAR	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Iterations	Time (secs.)	Iterations	Time (secs.)
100	200	10000	0 - 100	8	1.91	904	3.01	546	1.40
			0 - 1000	13	3.28	952	3.14	601	1.49
			0 - 10000	15	4.29	967	3.14	629	1.49
			0 - 100000	15	4.08	971	3.15	638	1.60
200	400	40000	0 - 100	10	8.71	2325	14.33	1430	8.18
			0 - 1000	13	13.06	2525	13.91	1717	8.48
			0 - 10000	16	35.44	2565	14.41	1741	8.42
			0 - 100000	15	19.84	2515	13.95	1725	8.46
300	600	90000	0 - 100	10	28.60	3947	35.94	2099	18.23
			0 - 1000	12	26.54	4588	35.94	2457	18.75
			0 - 10000	16	45.87	4633	38.34	2360	19.72
			0 - 100000	19	59.78	4452	33.78	2558	19.21
400	800	160000	0 - 100	10	50.14	5546	64.62	2850	35.27
			0 - 1000	11	54.57	6166	64.70	3630	34.83
			0 - 10000	17	85.02	6576	69.91	3844	40.37
			0 - 100000	21	143.54	6380	67.33	3905	45.03
TOTAL				221	584.67	56012	479.60	32730	268.93

Table 3. Comparison of RELAX, NETFLO, and NETSTAR on nxn Transportation Problems Having 10,000 Units of Supply

Problem Description				RELAX		NETFLO		NETSTAR	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Iterations	Time (secs.)	Iterations	Time (secs.)
100	200	10000	0 – 100	11	1.95	952	3.14	561	1.66
			0 – 1000	11	3.30	980	3.11	620	1.74
			0 – 10000	16	4.59	962	2.90	597	1.64
			0 – 100000	16	5.37	1000	3.06	665	1.80
200	400	40000	0 – 100	12	12.00	2257	13.25	1411	8.48
			0 – 1000	15	14.18	2607	15.58	1595	8.45
			0 – 10000	20	28.19	2509	16.52	1530	7.77
			0 – 100000	16	22.28	2443	13.39	1570	7.80
300	600	90000	0 – 100	11	37.95	3835	32.30	2107	18.85
			0 – 1000	13	28.98	4452	35.65	2489	21.09
			0 – 10000	17	43.69	4402	34.45	2625	21.27
			0 – 100000	19	65.23	4540	37.26	2529	20.24
400	800	160000	0 – 100	15	86.48	5421	65.19	2973	39.37
			0 – 1000	12	45.26	6378	69.38	3409	33.26
			0 – 10000	18	81.82	6343	71.20	3923	39.96
			0 – 100000	22	229.54	6248	71.31	3769	40.60
TOTAL				244	710.81	55329	487.69	31873	273.98

Table 4. Comparison of RELAX, NETFLO, and NETSTAR on nxn Transportation Problems Having 50,000 Units of Supply

Problem Description				RELAX		NETFLO		NETSTAR	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Iterations	Time (secs.)	Iterations	Time (secs.)
100	200	10000	0 - 100	9	1.77	822	2.55	527	1.47
			0 - 1000	14	3.04	863	2.74	632	1.67
			0 - 10000	14	4.07	1002	3.30	598	1.70
			0 - 100000	14	4.91	988	3.05	598	1.73
200	400	40000	0 - 100	9	9.29	2350	13.54	1326	7.14
			0 - 1000	15	14.95	2623	14.88	1667	9.00
			0 - 10000	19	25.61	2501	14.83	1567	8.20
			0 - 100000	21	48.93	2522	14.59	1571	8.92
300	600	90000	0 - 100	13	40.62	4057	37.22	2261	20.77
			0 - 1000	16	33.92	4363	35.13	2404	19.54
			0 - 10000	19	55.35	4867	40.48	2501	20.96
			0 - 100000	20	72.95	4560	37.79	2306	20.19
400	800	160000	0 - 100	15	98.38	5478	63.05	2951	34.74
			0 - 1000	13	58.12	6141	66.20	3276	32.92
			0 - 10000	21	97.21	6328	68.50	3575	37.96
			0 - 100000	23	180.73	6286	70.61	3635	40.95
TOTAL				255	747.85	55751	488.46	31395	258.86

Table 5. Comparison of RELAX, NETFLO, and NETSTAR on nxn Transportation Problems Having 100,000 Units of Supply

Problem Description				RELAX		NETFLO		NETSTAR	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Iterations	Time (secs.)	Iterations	Time (secs.)
100	200	10000	0 – 100	9	1.88	796	2.57	566	1.52
			0 – 1000	14	3.17	932	2.88	662	1.87
			0 – 10000	14	3.73	973	3.26	597	1.66
			0 – 100000	14	5.65	987	3.16	632	1.66
200	400	40000	0 – 100	11	9.76	2386	14.28	1207	6.56
			0 – 1000	14	15.42	2648	15.86	1673	8.67
			0 – 10000	16	21.57	2645	15.57	1669	9.03
			0 – 100000	19	24.06	2601	15.20	1694	8.61
300	600	90000	0 – 100	13	40.20	3948	34.73	2109	19.42
			0 – 1000	14	34.20	4389	36.46	2571	21.22
			0 – 10000	21	60.44	4557	39.36	2473	21.77
			0 – 100000	23	112.34	4501	37.01	2408	20.95
400	800	160000	0 – 100	17	109.23	5523	62.64	2925	31.90
			0 – 1000	13	58.18	6324	66.40	3593	38.27
			0 – 10000	24	125.40	6573	71.05	3590	40.45
			0 – 100000	20	151.76	6389	68.56	3598	40.26
TOTAL				256	742.79	56172	488.99	31967	273.82

Table 6. Comparison of NETSTAR and SAP_T on nxn Transportation Problems Having 1,000 Units of Supply

Problem Description				NETSTAR		SAP_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 - 100	515	1.47	38	0.82
			0 - 1000	603	1.66	37	1.38
			0 - 10000	575	1.38	36	1.40
			0 - 100000	582	1.60	30	1.40
200	400	40000	0 - 100	1601	8.50	85	3.51
			0 - 1000	1841	8.35	73	5.43
			0 - 10000	1836	9.45	61	6.20
			0 - 100000	1855	8.82	75	7.14
300	600	90000	0 - 100	2629	18.74	95	6.94
			0 - 1000	2789	19.90	90	11.18
			0 - 10000	2709	21.04	82	13.83
			0 - 100000	3184	23.28	83	13.17
400	800	160000	0 - 100	3021	32.18	118	13.18
			0 - 1000	4784	44.61	122	15.63
			0 - 10000	4615	43.16	115	23.38
			0 - 100000	4580	42.30	114	23.55
TOTAL				37719	286.30	1254	148.14

Table 7. Comparison of NETSTAR and SAP_T on nxn Transportation Problems Having 5,000 Units of Supply

Problem Description				NETSTAR		SAP_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 - 100	546	1.40	43	1.05
			0 - 1000	601	1.49	40	1.78
			0 - 10000	629	1.49	40	1.87
			0 - 100000	638	1.60	33	2.03
200	400	40000	0 - 100	1430	8.18	89	6.98
			0 - 1000	1717	8.48	90	10.48
			0 - 10000	1741	8.42	61	11.71
			0 - 100000	1725	8.46	78	12.98
300	600	90000	0 - 100	2099	18.23	92	16.10
			0 - 1000	2457	18.75	111	24.95
			0 - 10000	2360	19.72	91	32.08
			0 - 100000	2558	19.21	105	36.79
400	800	160000	0 - 100	2850	35.27	146	29.28
			0 - 1000	3630	34.83	161	41.85
			0 - 10000	3844	40.37	126	63.28
			0 - 100000	3905	45.03	148	65.24
TOTAL				32730	268.93	1454	358.42

Table 8. Comparison of NETSTAR and SAP_T on nxn Transportation Problems Having 10,000 Units of Supply

Problem Description				NETSTAR		SAP_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 – 100	561	1.66	46	1.25
			0 – 1000	620	1.74	40	1.94
			0 – 10000	597	1.64	39	1.89
			0 – 100000	665	1.80	33	1.89
200	400	40000	0 – 100	1411	8.48	96	7.58
			0 – 1000	1595	8.45	87	11.78
			0 – 10000	1530	7.77	59	12.92
			0 – 100000	1570	7.80	77	14.69
300	600	90000	0 – 100	2107	18.85	91	19.16
			0 – 1000	2489	21.09	110	27.04
			0 – 10000	2625	21.27	91	38.04
			0 – 100000	2529	20.24	99	43.56
400	800	160000	0 – 100	2973	39.37	138	32.11
			0 – 1000	3409	33.26	160	50.75
			0 – 10000	3923	39.96	127	77.58
			0 – 100000	3769	40.60	145	72.04
TOTAL				31873	273.98	1438	414.22

Table 9. Comparison of NETSTAR and SAP_T on nxn Transportation Problems Having 50,000 Units of Supply

Problem Description				NETSTAR		SAP_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 – 100	527	1.47	43	1.16
			0 – 1000	632	1.67	38	1.82
			0 – 10000	598	1.70	41	1.83
			0 – 100000	598	1.73	32	2.27
200	400	40000	0 – 100	1326	7.14	84	8.74
			0 – 1000	1667	9.00	85	12.96
			0 – 10000	1567	8.20	65	14.26
			0 – 100000	1571	8.92	78	16.59
300	600	90000	0 – 100	2261	20.77	90	24.05
			0 – 1000	2404	19.54	115	31.55
			0 – 10000	2501	20.96	91	46.88
			0 – 100000	2306	20.19	95	50.87
400	800	160000	0 – 100	2951	34.74	133	47.33
			0 – 1000	3276	32.92	164	65.38
			0 – 10000	3575	37.96	124	95.03
			0 – 100000	3635	40.95	156	91.55
TOTAL				31359	258.86	1434	512.17

Table 10. Comparison of NETSTAR and SAP_T on nxn Transportation Problems Having 100,000 Units of Supply

Problem Description				NETSTAR		SAP_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 - 100	566	1.52	43	1.30
			0 - 1000	662	1.87	39	1.82
			0 - 10000	597	1.66	41	1.86
			0 - 100000	632	1.66	32	2.06
200	400	40000	0 - 100	1207	6.56	91	8.24
			0 - 1000	1673	8.67	85	12.98
			0 - 10000	1669	9.03	65	13.50
			0 - 100000	1694	8.61	78	16.95
300	600	90000	0 - 100	2109	19.42	98	24.66
			0 - 1000	2571	21.22	117	31.25
			0 - 10000	2473	21.77	82	42.17
			0 - 100000	2408	20.95	92	54.03
400	800	160000	0 - 100	2925	31.90	136	40.96
			0 - 1000	3593	38.27	169	70.08
			0 - 10000	3590	40.45	124	95.86
			0 - 100000	3598	40.26	150	99.75
TOTAL				31967	273.82	1442	517.11

Table 11. Comparison of NETSTAR, SAP_T, and SAP_S_T on nxn Transportation Problems Having 1,000 Units of Supply

Problem Description				NETSTAR		SAP_T		SAP_S_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 – 100	515	1.47	38	0.82	48	1.04
			0 – 1000	603	1.66	37	1.38	25	1.28
			0 – 10000	575	1.38	36	1.40	25	1.34
			0 – 100000	582	1.60	30	1.40	25	1.33
200	400	40000	0 – 100	1601	8.50	85	3.51	100	4.06
			0 – 1000	1841	8.35	73	5.43	53	5.42
			0 – 10000	1836	9.45	61	6.20	49	5.89
			0 – 100000	1855	8.82	75	7.14	51	5.98
300	600	90000	0 – 100	2629	18.74	95	6.94	254	7.26
			0 – 1000	2789	19.90	90	11.18	113	11.73
			0 – 10000	2709	21.04	82	13.83	116	15.49
			0 – 100000	3184	23.28	83	13.17	112	15.30
400	800	160000	0 – 100	3021	32.18	118	13.18	321	13.43
			0 – 1000	4784	44.61	122	15.63	137	17.63
			0 – 10000	4615	43.16	115	23.38	146	22.46
			0 – 100000	4580	42.30	114	23.55	144	21.71
TOTAL				37719	286.30	1254	148.14	1719	151.35

Table 12. Comparison of NETSTAR, SAP_T, and SAP_S_T on nxn Transportation Problems Having 5,000 Units of Supply

Problem Description				NETSTAR		SAP_T		SAP_S_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 – 100	546	1.40	43	1.05	69	1.52
			0 – 1000	601	1.49	40	1.78	33	1.63
			0 – 10000	629	1.49	40	1.87	32	1.56
			0 – 100000	638	1.60	33	2.03	32	1.56
200	400	40000	0 – 100	1430	8.18	89	6.98	128	6.74
			0 – 1000	1717	8.48	90	10.48	65	6.96
			0 – 10000	1741	8.42	61	11.71	58	8.70
			0 – 100000	1725	8.46	78	12.98	58	8.85
300	600	90000	0 – 100	2099	18.23	92	16.10	213	19.21
			0 – 1000	2457	18.75	111	24.95	94	18.60
			0 – 10000	2360	19.72	91	32.08	103	19.47
			0 – 100000	2558	19.21	105	36.79	97	20.19
400	800	160000	0 – 100	2850	35.27	146	29.28	187	27.44
			0 – 1000	3630	34.83	161	41.85	86	34.57
			0 – 10000	3844	40.37	126	63.28	80	38.30
			0 – 100000	3905	45.03	148	65.24	85	39.41
TOTAL				32730	268.93	1454	358.42	1420	254.71

Table 13. Comparison of NETSTAR, SAP_T, and SAP_S_T on nxn Transportation Problems Having 10,000 Units of Supply

Problem Description				NETSTAR		SAP_T		SAP_S_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 – 100	561	1.66	46	1.25	73	1.68
			0 – 1000	620	1.74	40	1.94	36	1.43
			0 – 10000	597	1.64	39	1.89	32	1.33
			0 – 100000	665	1.80	33	1.89	32	1.31
200	400	40000	0 – 100	1411	8.48	96	7.58	152	6.95
			0 – 1000	1595	8.45	87	11.78	71	7.88
			0 – 10000	1530	7.77	59	12.92	70	7.89
			0 – 100000	1570	7.80	77	14.69	72	7.74
300	600	90000	0 – 100	2107	18.85	91	19.16	239	24.06
			0 – 1000	2489	21.09	110	27.04	93	21.29
			0 – 10000	2625	21.27	91	38.04	107	25.98
			0 – 100000	2529	20.24	99	43.56	105	26.74
400	800	160000	0 – 100	2973	39.37	138	32.11	278	39.89
			0 – 1000	3409	33.26	160	50.75	117	43.80
			0 – 10000	3923	39.96	127	77.58	120	47.04
			0 – 100000	3769	40.60	145	72.04	117	48.33
TOTAL				31873	273.98	1438	414.22	1714	313.34

Table 14. Comparison of NETSTAR, SAP_T, and SAP_S_T on nxn Transportation Problems Having 50,000 Units of Supply

Problem Description				NETSTAR		SAP_T		SAP_S_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 – 100	527	1.47	43	1.16	74	1.96
			0 – 1000	632	1.67	38	1.82	25	1.70
			0 – 10000	598	1.70	41	1.83	32	1.60
			0 – 100000	598	1.73	32	2.27	32	1.62
200	400	40000	0 – 100	1326	7.14	84	8.74	151	8.89
			0 – 1000	1667	9.00	85	12.96	67	8.96
			0 – 10000	1567	8.20	65	14.26	71	9.55
			0 – 100000	1571	8.92	78	16.59	70	10.23
300	600	90000	0 – 100	2261	20.77	90	24.05	255	27.60
			0 – 1000	2404	19.54	115	31.55	102	25.27
			0 – 10000	2501	20.96	91	46.88	101	24.87
			0 – 100000	2306	20.19	95	50.87	101	26.02
400	800	160000	0 – 100	2951	34.74	133	47.33	351	45.69
			0 – 1000	3276	32.92	164	65.38	128	47.42
			0 – 10000	3575	37.96	124	95.03	132	57.98
			0 – 100000	3635	40.95	156	91.55	123	57.28
TOTAL				31395	258.86	1434	512.17	1815	356.64

Table 15. Comparison of NETSTAR, SAP_T, and SAP_S_T on nxn Transportation Problems Having 100,000 Units of Supply

Problem Description				NETSTAR		SAP_T		SAP_S_T	
Size	Nodes	Arcs	Cost Range	Iterations	Time (secs.)	Augmenting Paths	Time (secs.)	Augmenting Paths	Time (secs.)
100	200	10000	0 – 100	566	1.52	43	1.30	77	1.99
			0 – 1000	662	1.87	39	1.82	25	1.68
			0 – 10000	597	1.66	41	1.86	31	1.65
			0 – 100000	632	1.66	32	2.06	31	1.68
200	400	40000	0 – 100	1207	6.56	91	8.24	160	9.46
			0 – 1000	1673	8.67	85	12.98	69	9.50
			0 – 10000	1669	9.03	65	13.50	65	9.37
			0 – 100000	1694	8.61	78	16.95	68	9.55
300	600	90000	0 – 100	2109	19.42	98	24.66	256	29.17
			0 – 1000	2571	21.22	117	31.25	103	26.19
			0 – 10000	2473	21.77	82	42.17	98	25.59
			0 – 100000	2408	20.95	92	54.03	99	25.65
400	800	160000	0 – 100	2925	31.90	136	40.96	352	47.87
			0 – 1000	3593	38.27	169	70.08	136	51.90
			0 – 10000	3590	40.45	124	95.86	131	56.40
			0 – 100000	3598	40.26	150	99.75	133	59.85
TOTAL				31967	273.82	1442	517.11	1834	367.50